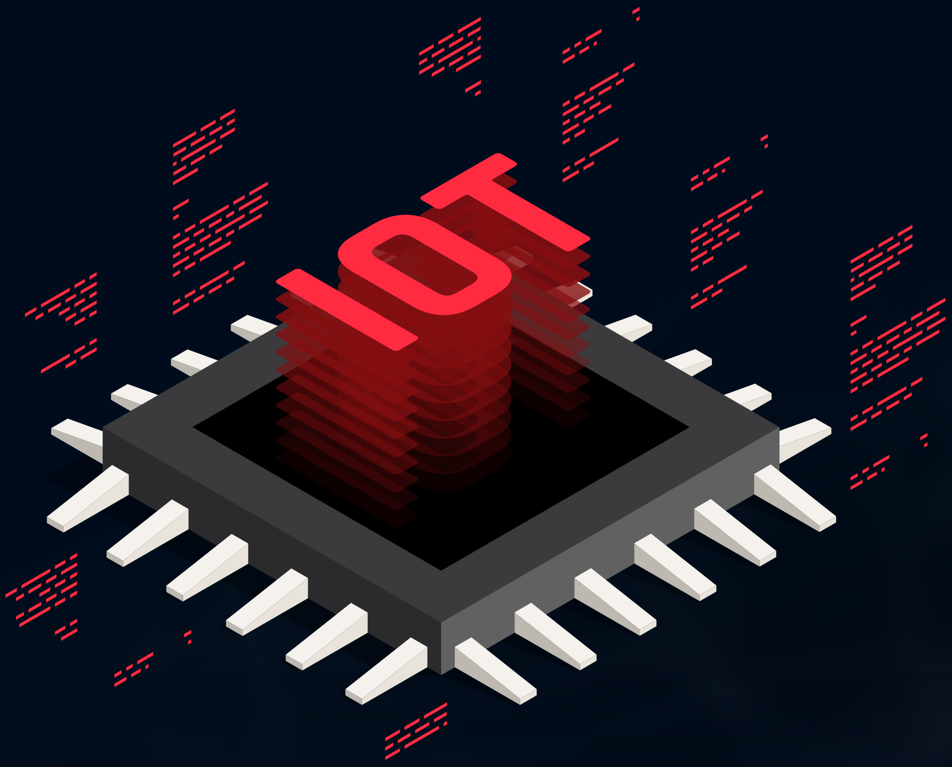


Hands-on Internet of Things Hacking

Copyright © 2021 Payatu Consulting Pvt. Ltd. All Rights Reserved.

Payatu

Hands-on IoT Hacking E-book

IoT Security MasterClass E-Book

Copyright notice:
This e-book and its content is copyright of Payatu Consulting Pvt. Ltd.
Copyright © 2021 Payatu Consulting Pvt. Ltd. All Rights Reserved.

Any redistribution or reproduction of part or all of the contents in any form is prohibited other than the following:

- You may print or download to a local hard disk extracts for your personal and non-commercial use only
- You may copy the content to individual third parties for their personal use, but only if you acknowledge the e-book as the source of the material

You may not, except with our express written permission, distribute or commercially exploit the content. Nor may you transmit it or store it in any other website or other form of electronic retrieval system.

Copyright © 2021 Payatu Consulting Pvt. Ltd. All Rights Reserved.

Table of contents

Table of contents

About Payatu....12
Payatu Consulting and Training....12
Conferences....15
    About Nullcon
    About hardwear.io
Payatu Product Offerings....15
    About Cloudfuzz
    About EXPLIoT
        EXPLIoT Offerings
        EXPLIoT Products
            1. EXPLIoT IoT Security learning kit
            2. Bus Auditor
            3. EXPLIoT Nano
            4. DIVA Board
            5. ZigBee AuditorPreface
Preface....19
The Authors....20
    Aseem Jakhar
    Asmita Jha
    Shakir Zari
    Dattatray Hinge
    Appar Thusoo
    Munawwar Hussain Shelia
    Arun Magesh
Part 1- Introduction....21
Chapter 1: IoT Introduction And Architecture....22
    1. IoT != Hardware
    2. Introduction
        2.1 What is IoT?
        2.2 Where is IoT used?
    3. IoT Architecture
        3.1 High level view
        3.2 Functional Architecture
        3.3 Layered model
Chapter 2: IoT Attack Surface....28
    1. Mobile
    2. Cloud
    3. Device

- 4. Communication
- 5. Web
- 6. Others
- 7. Device Attack Surface
  - 7.1 Storage
    - 7.1.1 SD Card
    - 7.1.2 USB
    - 7.1.3 Non-volatile Memory
    - 7.1.4 Volatile Memory
    - 7.1.5 Microcontroller Internal Memory
  - 7.2 Hardware Communication Interface
    - 7.2.1 UART
    - 7.2.2 Microcontroller Debug Port
    - 7.2.3 I2C
    - 7.2.4 SPI
    - 7.2.5 USB
  - 7.3 Sensor
  - 7.4 Human Machine Interface
  - 7.5 Other Hardware interfaces
- 8. Network Communication Interface
  - 8.1 Wi-Fi
  - 8.2 Ethernet
  - 8.3 Radio

Chapter 3: IoT Top Ten Vulnerabilities......40

- OWASP Internet Of Things Top Ten Vulnerabilities 2014
- Payatu Internet of Things Top Ten Vulnerabilities 2018
- P1. Hardcoded Sensitive information
- P2. Enabled hardware debug ports
- P3. Insecure Firmware
- P4. Insecure Data storage
- P5. Insufficient Authentication
- P6. Insecure Communication
- P7. Insecure Configuration
- P8. Insufficient data input filtration
- P9. Insecure Mobile Interface
- P10. Insecure Cloud/Web Interface

Part 2- Protocols......48

MQTT

Chapter 4: Introduction To MQTT Protocol And Security......49

- 1. Introduction
  - 1.1 Use and Popularity
  - 1.2 Architecture Overview

2. MQTT Communication
    2.1 Communication Overview
    2.2 Communication Example
3. MQTT Protocol Intro
    3.1 MQTT Control Packets Intro
    3.2 Quality of Service (QoS)
    3.3 Packets and Usage
        3.3.1 CONNECT
        3.3.2 CONNACK
        3.3.3 PUBLISH
        3.3.4 SUBSCRIBE
        3.3.5 SUBACK
        3.3.6 UNSUBSCRIBE
        3.3.7 UNSUBACK
4. MQTT Security
    4.1 Tools
    4.2 Recon
    4.3 Analysis
    4.4 Attacks
        4.4.1 Denial of Service via Duplicate Client ID
        4.4.2 Client ID used for Authentication/Access Control
        4.4.3 Cloning the Client
        4.4.4 Attacking the Application via Malicious Input
        4.4.5 Accessing and Manipulating Devices via messages
    4.5 Mitigations
5. Conclusion

Chapter 5: MQTT Broker Security - 101..........63
1. Introduction
2. Data and Connection Security
    2.1 Secure Connection
        2.1.1 Mosquitto Broker
        2.1.2 AWS IoT Core
    2.2 Application data or MQTT Payload security
3. Client Authentication
    3.1 Client Identifier (Client Id)
        3.1.1 Mosquitto Broker
        3.1.2 AWS IoT Core
    3.2 Username and Password
        3.2.1 Mosquitto Broker
        3.2.1.1 Allow Anonymous
        3.2.1.2 Password file Path
        3.2.2 AWS IoT Core

- 3.3. Client SSL Certificate
    - 3.3.1 Mosquitto Broker
    - 3.3.1.1 Client Certificate require
    - 3.3.1.2 Root CA file Path
    - 3.3.1.3 Server Certificate and keyfile Path
    - 3.3.2 AWS IoT Core
- 4. Restrict access to server resources
    - 4.1 Mosquitto Broker
        - 4.1.1 topic name
        - 4.1.2 user name
        - 4.1.3 pattern substitution with the topic
        - 4.1.3.1 Client ID substitution
        - 4.1.3.2 User name substitution
    - 4.2 AWS IoT Core
- 5. Certificate Revocation List
    - 5.1 Mosquitto Broker
    - 5.2 AWS IoT Core
- 6. Conclusion

COAP

Chapter 6: Introduction To CoAP Protocol And Security..73

- 1. Introduction
    - 1.1 CoAP Features
    - 1.2 HTTP Lineage
    - 1.3 Use and Popularity
- 2. CoAP Communication
    - 2.1 Protocol Overview
        - 2.1.1 CoAP Messages
        - 2.1.2 CoAP Request/Response
        - 2.1.3 CoAP Request and Response Codes
        - 2.1.4 CoAP Options
    - 2.2 Discovery Mechanism
- 3. CoAP Security
    - 3.1 Protocol Security/Authentication
    - 3.2 Tools
    - 3.3 Recon
    - 3.4 Analysis
    - 3.5 Attacks
        - 3.5.1 Path Traversal (Old wine in new bottles)
        - 3.5.2 Cross Protocol Attacks
        - 3.5.3 Attacking the Application via Malicious Input
        - 3.5.4 Accessing and Manipulating Devices Via Requests
        - 3.5.5 Cloning or MitM'ing the Server (Sensor)
        - 3.5.6 Distributed Denial of Service using CoAP

        3.6 Mitigations
    4. Conclusion

BLE

Chapter 7: Bluetooth Low Energy – 101..88
    1. Generic Access Profile (GAP)
    2. Generic Attribute (GATT)
        2.1 Profile
        2.2 Services
        2.3 Characteristics
        2.4 Connecting your Bluetooth dongle:
        2.5 Scanning for Bluetooth devices
        2.6 Reading and writing data

ZIGBEE

Chapter 8: ZigBee Protocol – 101..97
    1. History
    2. ZigBee Overview
    3. IEEE 802.15.4 Protocol
        3.1 PHY Layer
        3.2 MAC Layer
    4. IEEE 802.15.4 Network Model
        4.1 Node Type
        4.2 Full-Function Device (FFD)
        4.2 Reduced-Function Devices (RFD)
        4.4 Topologies
        4.5 Star
        4.6 Peer–to–Peer
        4.7 Node Addressing Modes
    5. ZigBee Protocol
        5.1 Application Layer
        5.2 Application Support Sub-Layer (APS)
        5.3 APS data entity (APS-DE)
        5.4 APS management entity (APM-SE)
        5.5 Application Framework
        5.6 Application Profiles
        5.7 Clusters
        5.8 ZigBee Device Objects (ZDO)
    6. Network Layer
        6.1 Network Layer Data Entity (NLDE)
        6.2 Network Layer Management Entity (NLME)
        7. ZigBee Device Type
            7.1 ZigBee Coordinator (ZC)
            7.2 ZigBee Router (ZR)
            7.3 ZigBee End Device (ZED)

8. Network Topology
    8.1 Star
    8.2 Tree
    8.3 Mesh
9. Addressing in ZigBee
    9.1 Device Addressing
    9.2 IEEE Address
    9.3 Network Address
    9.4 ZigBee Network Identity
    9.5 PAN Identifier (PAN ID)
    9.6 Extended PAN ID (EPID)
    9.7 Application level addressing
        9.7.1 End Point Address
        9.7.2 Cluster Identifier (Cluster ID)
10. Messaging in ZigBee
    10.1 Broadcast
    10.2 Unicast
    10.3 Group Multicast
    10.4 Inter-PAN Communication
11. Conclusion
Chapter 9: ZigBee Security - 101......108
1. ZigBee Security Architecture (Open Trust)
2. Trust Center
3. Security Modes in ZigBee
    3.1 Distributed Security Mode
    3.2 Centralized Security Mode
4. ZigBee Security Keys
    4.1 Network Key
    4.2 Link Key
5. Key Management
6. ZigBee Protocol Security
    6.1 Auxiliary Security Header
    6.2 Security control
    6.3 Frame counter
    6.4 Source address
    6.5 Key sequence number
ZigBee Vulnerabilities
7. Implementation vulnerabilities
    7.1 Security keys stored insecurely
    7.2 Over-The-Air insecure key transportation
    7.3 Energy Depletion Attack
    7.4 Invalid security header
    7.5 Polling Rate

8. Protocol vulnerabilities
    8.1 Network Jamming Vulnerability
        8.1.1 Radio Jamming
        8.1.2 Link Layer Jamming
        8.1.3 Link key vulnerability
        8.1.4 Default Link Key
        8.1.5 Unencrypted Link-Key
        8.1.6 Re-using Link key
    8.2 Unauthenticated ACK frame vulnerability
        8.2.1 ACK Spoofing
        8.2.2 ACK Dropping
        8.3 Replay-protection vulnerability
9. Conclusion

Part 3- RADIO......118

Chapter 10: Introduction to Software Defined Radio: Hardware tools......119

1. Introduction
2. Prerequisite
3. Tools of trade:
    Hardware
    3.1 Realtek SDR Dongle (RTL820T2)
    3.2 HackRF One
    3.3 BladeRF 2.0 micro
    3.4 USRP B210
4. Antennas
5. Conclusion

Chapter 11: Introduction to Software Defined Radio: Software tools and Approach techniques......125

1. Introduction
2. Software
3. Recon tools
    3.1 Basic assessment
    3.2 Advanced assessment
    3.3 Other Points of Interest
4. How to approach a target
    4.1 Recon
    4.2 Seek
    4.3 Record
    4.4 De-Modulation
    4.5 Process
    4.6 Decode
    4.7 Attack!
5. Conclusion

Part 4- Firmware....133
Chapter 12 : Firmware Reverse Engineering....134
1. Bare Metal Firmware
2. Fully Fledged Operating System
3. Security Tools
    3.1 Binwalk
    3.2 Qemu
    3.3 Gdb-multiarch
    3.4 Firmware ModKit
4. Conclusion
Part 5- Hardware....139
Chapter 13: Introduction to Hardware Recon....140
1. Importance of Hardware Recon
2. Hardware Tools
3. Basic knowledge of electronics
4. Different types of packages of components
    4.1 Through-hole mount package
    4.2 Surface Mount Package
    4.3 Types of SMD components
5. Printed Circuit Board (PCB)
6. FCC ID
7. Disassembly
    7.1 PCB Board Analysis
    7.2 Chip Analysis
        7.2.1 WINBOND
        7.2.2 BROADCOM
        7.2.3 FR9886
        7.2.4 MXIC
8. Serial Flash Recon
    8.1 Interfacing IC with our computer
9. Pinouts (headers)
10. Conclusion
Chapter 14: Introduction to and Identification of Hardware Debug Ports....156
1. Why do PCBs have to debug ports?
2. Bus Auditor
    2.1 Features
3. DIVA
4. JTAG (Joint Test Action Group)
    4.1 Identify JTAG Port on DIVA
    4.2 Bus Auditor Result
5. SWD (Serial Wire Debugger)
    5.1 Identify SWD Port on DIVA
5.2 Bus Auditor Result

6. I2C(Inter-Integrated Circuit)
    6.1 Identify I2C on the DIVA board
    6.2 Bus Auditor Result
7. UART (Universal Asynchronous Receiver-Transmitter)
    7.1 Identify UART on DIVA
    7.2 Bus auditor Result
        Bus Auditor
8. Conclusion

SPI

Chapter 15: Hardware Attack Surface- SPI....170
1. Introduction
2. Possible Attack Scenarios
3. Attacking hardware via SPI
    3.1 Recon
        Case 1: You do not have the actual hardware with, but you know, the FCCID number of the device.
        Case 2: You got access to the hardware.
    3.2 Sniffing SPI Communication
    3.3 Extracting data/firmware from SPI Memory Chip
        Case 1: You are successful in finding the tools or framework that can support your SPI memory chip
        Case 2: You cannot find any tool/framework that can support your SPI memory chip
4. Measures that can mitigate the attacks
5. Conclusion

I2C

Chapter 16: Hardware Attack Surface- I2C....181
1. Introduction
2. Possible Attack Scenarios
3. Attacking hardware via I2C
    3.1 Recon
        Case 1: You do not have the actual hardware, but you know the FCCID number of the device.
        Case 2: You got access to the hardware.
    3.2 Sniffing I2C Communication
    3.3 Extracting data from I2C memory Chip.
        Case 1 You are successful in finding the tools and framework that can support your I2C memory chip
        Case 2: You cannot find any tool/framework that can support your I2C memory chip
4. What measures can make it difficult for an attacker to attack?
5. Conclusion

UART

Chapter 17: Hardware Attack Surface- UART..192
1. Introduction
2. Possible Attack Scenarios
3. Attacking hardware via UART interface
    3.1 Recon
        Case 1: You do not have the actual hardware, but you know the FCCID number of the device.
        Case 2: You got access to the hardware.
    3.2 UART port interfacing, communication sniffing & device accessing
    3.3 Accessin
4. Conclusion

JTAG/SWD

Chapter 18: Hardware Attack Surface- JTAG, SWD..200
1. Introduction
    1.1 JTAG
    1.2 SWD
2. Possible Attack Scenarios
3. Performing the Attack
    3.1 Recon
        Case 1: You do not have the actual hardware, but you know the FCCID number of the device.
        Case 2: You got access to the hardware.
    3.2 Sniff the communication
    3.3 Interfacing
4. Conclusion

Side-Channel Analysis

Chapter 19: Introduction To Side-Channel Attacks (SCA)...212
1. Introduction
2. Types of side channel attacks
    2.1 Power Attacks
        2.1.1 Simple power analysis (SPA)
        2.1.2 Differential power analysis (DPA)
        2.1.3 Correlation Power Analysis (CPA)
    2.2 Electromagnetic (EM) Attacks
    2.3 Timing Attacks
    2.4 Cache Attacks
    2.5 Differential Fault Attacks
    2.6 Acoustic Attacks
3. Why side channel attacks are so alarming
4. Possible countermeasures for side channel attacks
5. Conclusion

Fault Injection

Chapter 20: Introduction To Fault Injection Attack (FI)........217
- 1. Fault injection techniques
    - 1.1 Clock glitch
    - 1.2 Voltage glitch
    - 1.3 EM glitch
    - 1.4 Optical injection
- 2. Examples of fault injection attacks
    - CVE-2019-15894
    - CVE-2020-15048
    - CVE-2020-13468
    - CVE-2020-15808
- 3. Countermeasures
- 4. Conclusion

Part 6- Miscellaneous........224

Chapter 21: Famous IoT Attacks & Vulnerabilities........225
- 1. Ripple20
    - CVE-2020-11896
    - CVE-2020-11898
    - CVE-2020-11901
- 2. Meltdown and Spectre
- 3. Mirai botnet attack on IoT devices
- 4. Rolljam Attack
- 5. Sweyntooth vulnerabilities
- 6. Philips Hue smart lightbulb Zigbee vulnerability
- 7. Fault Injection Attack on ESP32
- 8. BLESA Attack
- 9. Bluetooth attack on Tesla Model X
- 10. Amnesia:33
- 11. Conclusion

Summary........233

About Payatu

Payatu is a research focused security testing service organization specialized in IoT and embedded products, web, mobile, cloud and infrastructure security assessments, and in-depth technical security training. Our state-of-the-art research, methodologies, and tools ensure the safety of our client's assets.

At Payatu, we believe in following one's passion, and with that thought, we have created a world-class team of researchers and executors who are bending the rules to provide the best security services. We are a passionate bunch of folks working on the latest and leading-edge security technology.

Payatu’s impact is delivered every day through our 3 verticals.

1. Consulting and Training
2. Conferences.
3. Products

Payatu Consulting and Training

Payatu is a Research Focused, CERT-In impaneled Cybersecurity Consulting company specializing in security assessments of IoT product ecosystem, Web application & Network with a proven track record of securing applications and infrastructure for customers across 20+ countries.

We have delivered talks and workshops worldwide in some of the world's top security conferences .

Our Services

1. **IoT Security Testing**
    - Extensive/comprehensive testing of all IoT products.
    - Prevent and combat security vulnerabilities related to hardware, firmware, mobile apps, cloud, and others.

2. **Red Team Assessment**
    - Measure how your systems, applications, and security controls can withstand online and offline risks
    - Get counter measures suitable for all your security needs

3. **Product Security**
   - Ensure that you save time and effort in security bug fixing and patching cycle
   - Mitigate potential threats to your products
   - Completely secure your products

4. **AI-ML security design**
   - AI Security Assessment</h3>
   - Secure your AI application against esoteric and potentially severe security and privacy threats.

5. **Web security testing**
   - Extensive audit techniques sweep every corner of your system to discover potential attack surfaces
   - Dual security audit execution approach, i.e. automated and manual security is followed

6. **Mobile Security Testing**
   - Intensive analysis of Android and iOS mobile applications for security vulnerability and possible weak spots
   - Manual testing of each component of a mobile application rather than an automated vulnerability scanner

7. **DevSecOps**
   - An all-inclusive approach is taken to incorporate security as an integral component of the entire delivery pipeline from the start
   - DevSecOps platforms are tailored to meet your unique criteria

8. **Code Review**
   - Source code security audits provided for both thick client and thin client applications.
   - A combination of manual code review and automated code analysis tools are deployed

9. **Cloud Security**
   - Review and assess applications deployed in your cloud for security and design flaws
   - Get configuration review of your cloud platform done

10. **Critical Infrastructure**
    - Critical Infrastructure
    - Extensive assessment of critical infrastructures like power plants and hospitals
    - Evaluation of individual components of the critical infrastructure

Training

1. **Practical IOT Hacking**
    - Get an overall understanding of the entire IoT technology suite, including IoT protocols and sensors, along with their underlying weaknesses
    - The practical labs enable attendees to identify and exploit vulnerabilities in IoT

2. **Web application Security**I
    - dentify and exploit vulnerabilities that are found in a myriad of web applications on the Internet
    - Get a hands-on approach to chain multiple vulnerabilities to fulfil the objectives of gaining access to data or taking over the underlying operating system

3. **Mobile Application Security**
    - Get familiarized with comprehensive technical explanation of some of the most common mobile-based vulnerabilities, as well as how to verify and exploit them
    - Learn how to bypass both Android and iOS security models

4. **Cloud Security Training**
    - Learn to break into applications and services running in famous cloud platforms like AWS and Azure cloud
    - Learn about protecting your application deployed in the cloud from an offensive perspective

5. **Windows Kernel Security**
    - Go through the fundamentals of Windows Kernel internals, as well as learn fuzzing of Windows Kernel Mode drivers
    - Learn how to exploit different vulnerabilities present in Windows Kernel

6. **Network Infrastructure Security**
    - Learn to perform advanced recon and pwn enterprise by exploiting vulnerabilities in the domain controller, application servers, database servers and network devices, like a pro
    - Go to the next level of taking over networks and servers by using your own custom exploit codes, other than Metasploitl

7. **Practical DevSecOps**
    - Learn what is DevSecOps and how to integrate it in your infrastructure
    - Learn how to use static analysis (SAST), Dynamic Analysis (DAST), Infrastructure security assessment for implementing secure SDLC suited for your organization

8. **Attack Monitoring for SOC**
    - Learn how to take control of enterprise-wide logs and analyze them in real-time for security monitoring and alert, using ELK framework
    - Learn to scale the Elastic Stack and generate powerful visualization & data modelling for your organisation, using Kibana

9. **AI/ML Security**
    - Understand application of ML/AI in security product development
    - Learn about security issues in ML/AI model like model stealing and poisoning attacks.

Conferences

Payatu runs Asia's most respected cybersecurity conference, Nullcon along with Hardwear.io, a hardware security-focused conference.

About Nullcon -

nullcon security conference is an annual security event and the largest information security conference held in Goa, India. After years of efforts put in the event, it has now become a world-renowned platform to showcase the latest and undisclosed research.

About hardwear.io -

Hardware security conference is an annual hardware security event held in The Hague, Netherlands. The first of its kind, it is being organized to answer emerging threats and attacks on hardware. We aim to make it the largest platform, where hardware security innovation happens.

Payatu Product Offerings

Payatu has also released two products in private beta trials. Cloudfuzz, an advanced, fuzzing framework and EXPLIoT, a firmware analysis framework,

About Cloudfuzz

Cloudfuzz is a platform that lets you test your code for bugs by running your software with millions of test cases , a technique called Fuzzing. It's an intelligent, self-sufficient and scalable system that only needs configuration once. Helps you by generating reports for bugs encountered while Fuzzing.

About EXPLIoT

EXPLIoT is an open-source framework for security testing and exploiting IoT. It is an initiative by Payatu, a research-powered cybersecurity company with expertise in IoT, Embedded, Mobile, Cloud security, and vulnerability research. More details – https://payatu.com. We also organize one of the largest information security conferences in Asia called Nullcon, Goa, India – https://nullcon.net/ and one of the most active Hardware Security conferences in Europe known as hardwear.io, The Hague, Netherlands – http://hardwear.io. For more details about EXPLIoT and our products, visit https://expliot.io/

EXPLIoT Offerings

1. IoT Auditor – A comprehensive IoT Security and Compliance Assessment Platform capable of performing firmware, hardware, network, radio and cloud analysis. The Community Edition is FREE and includes basic Firmware Auditor, the Cloud based firmware analysis module of IoT Auditor. More Details- https://expliot.io/pages/firmware-auditor
2. **Online Store -** Our online store where you can purchase hardware and radio tools for IoT Security. More details- https://expliot.io/collections/frontpage
3. **Practical IoT Hacking Training -** We regularly deliver the training and small workshops at all premier security conference such as BlackHat, Defcon, Brucon, Hack in Paris, Zerocon, etc. More details - https://expliot.io/pages/training
4. **EXPLIoT Framework** - An open-source IoT Security Testing and Exploitation Framework. More details - https://gitlab.com/expliot_framework/expliot

EXPLIoT Products

1. EXPLIoT IoT Security learning kit

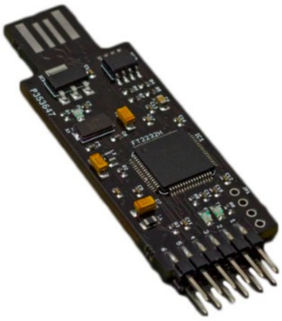
The EXPLIoT IoT Security learning kit is a great way to get started with IoT device security and exploitation using the DIVA (Damn Insecure and Vulnerable Application) board and other targets. This kit includes everything that you need in your arsenal to perform UART, I2C, SPI, JTAG, ZigBee, BLE analysis. This Lab Manual provides guidance and step by step process of performing each lab. It will teach you everything from finding the communication ports, sniffing, manipulating communication to exploiting IoT devices.

2. Bus Auditor

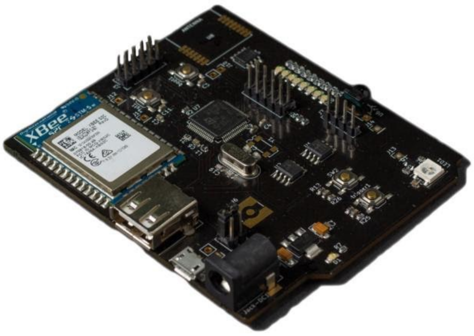
BUS Auditor is a compact multi-protocol tool used for scanning and identifying debugging and communication interfaces exposed on any hardware board. It can brute force several hardware protocols, including JTAG, arm SWD, UART, and I2C.The device has 16 channels and every channel can be used to interface with a pin-out on the target board.

3. EXPLIoT Nano

EXPLIoT-NANO is a compact hacker-friendly multi-purpose, multi-protocol hardware tool mainlyused to debug and program microcontrollers/ processors and flash chips.EXPLIoT-NANO can be configured to support hardware protocols including, UART, I2C, SPI, ARM SWD, and JTAG. Even though it runs on 3.3V, all I/O pins are 5V tolerant.



4. DIVA Board



DIVA [Damn Insecure and Vulnerable Application] board is a connected IoT device and a vulnerable target board designed to teach the basics of IoT security. DIVA integrates an ARM cortex M4 microcontroller and a 802.15.4 radio on the board. It comes with many more on-board peripherals including, SPI and I2C EEPROMS, IR receiver, temperature sensor, RGB LED, input switches, etc. The board provides a standard JTAG debug interface as well as a SWD port that can be used to debug programs from the host PC. The inbuilt USB port can be used for accessing the serial console and for firmware upgrades in DFU mode.

4. DIVA Board

DIVA [Damn Insecure and Vulnerable Application] board is a connected IoT device and a vulnerable target board designed to teach the basics of IoT security. DIVA integrates an ARM cortex M4 microcontroller and a 802.15.4 radio on the board. It comes with many more on-board peripherals including, SPI and I2C EEPROMS, IR receiver, temperature sensor, RGB LED, input switches, etc. The board provides a standard JTAG debug interface as well as a SWD port that can be used to debug programs from the host PC. The inbuilt USB port can be used for accessing the serial console and for firmware upgrades in DFU mode.

5. ZigBee Auditor

Zigbee Auditor
ZBA-XA rev 1.0
J2

ZigBee Auditor is a tool developed for professionals working with the ZigBee network asdevelopers, auditors, and cybersecurity professionals. ZigBee Auditor - XA comes with an onboardantenna that provides an indispensable 100m signal range for network auditing andscanning task. To use ZigBee Auditor, you need to install EXPLIoT, an open-source framework forsecurity testing and exploiting IoT. ZigBee Auditor provides ZigBee network scanning, packet sniffing, and packet replay functionality.

Preface

This E-book has been curated for cybersecurity professionals looking to take the first step into the IoT security domain. Our expert authors have written each chapter in fine detail to help you understand and educate you on the core fundamentals of IoT security. Our objective is to create a singular knowledge base which is easy-to-access and comprises of everything cybersecurity beginners would need to equip themselves and strive forward in the industry. We would like to thank the authors and the editing team for putting their time and effort to help turn this idea of an e-book into a reality.

The Authors

Aseem Jakhar
Director of
Security Research
and Development

Munawwar Hussain Shelia
IoT Security Researcher

Appar Thusoo
IoT Security Consultant

Asmita Jha
IoT Security Consultant

Shakir Zari
Hardware Security Researcher

Arun Magesh
IoT Security Researcher

Dattatray Hinge
Lead Developer

Part 1

INTRODUCTION

Chapters: 1 to 3

What to expect:

A holistic overview of the IoT ecosystem encapsulating its architecture, usage, attack surfaces, along with Payatu's own list of Top 10 vulnerabilities.

Chapter 1: IoT Introduction And Architecture

We have been getting a lot of queries recently from folks interested in Internet of Things security about where to start in IoT security research so, we decided to create an e-book to help researchers who want to get into IoT security, IoT penetration testing, and exploitation. The problem with every new and complex technology for security researchers is not knowing where to start and how/where to attack. This is a common problem and has a common solution i.e. breaking the technology into small components and start learning each component individually. This process makes you master each component and guides you to focus on the most interesting components, based on your area of interest. If you have read till here, we are assuming you are going to stick around and read through. So, without any delay, let's start!

Note:

1. The information in this e-book is generic and can be applied to the security research of IoT products in any domain irrespective of their usage, including Home automation, Industrial Control Systems, Healthcare, Transportation, etc.
2. We will use the words device, hardware, and sensor interchangeably to mean the same thing unless specifically mentioned with an explanation.
3. We mention IoT ecosystem to mean an IoT product or a solution due to the nature of the IoT technology that comprises of different technologies.

1. IoT != Hardware

This is a common misconception among folks that IoT means hardware only, which creates an imaginary roadblock and discourages most security researchers from venturing into IoT security. Yes, there is hardware involved, and the skills required to analyze it can be learnt. IoT security research is not difficult, but requires dedication and inclination to learn. As you read through this chapter, you will realize that the hardware only forms 1/3 part of the IoT ecosystem. On top of that, if you can compromise other components (for ex. Cloud), you can cause more damage than just hacking into the device. We had the same inhibition a few years ago when we started working on IoT security, so we broke down the problem into small pieces and attacked them individually, and learned some cool tricks along the way.

2. Introduction

2.1 What is IoT?

There are many definitions of IoT on the Internet, and the crux of mastering technology is to understand the basic ideology behind it, which helps in defining your own meaning and applicability of the same. Everyone can have their own definition, and for us, IoT is all about three important things.

1. **Automation:** Let's face it, we are lazy, and the future is all about making us lazier and automating any task that we do manually.
2. **Virtual-Physical world interface:** Creating a bridge between the physical and the virtual world. In simple terms allow the virtual world to read and write from/to the physical world. When I say read, I mean sensing the physical environment and converting the state into data and sending it across to the virtual data storage for further analysis such as temperature sensors, medical sensors, cameras, etc. Write means to control the physical world with an action i.e., converting the data into an action on the physical world such as door locks, controlling the vehicle operations, spraying water, medical pumps, etc. You get the idea.
3. **Insight and Decision making:** The data gathered from the devices can be analyzed in realtime to understand the environment better, act on certain events, find the root cause of any physical world problems, etc.

The IoT technology thus empowers both end-users as well as the vendors with real-time information and automation of the tasks at hand.

Based on the above definition, if we were to create a technology to solve this problem, we would require

1. hardware device that provides the virtual-physical interface
2. A backend data storage to store and computing power to perform statistical analysis on the data.
3. A virtual interface for the users to view the analyzed data and send commands to the physical world.

The first is solved by having economical hardware devices embedded with the respective sensors/controllers, the second is conveniently solved by the cloud, and finally, the third is easily solved by a mobile application and/or a web application.

2.2 Where is IoT used?

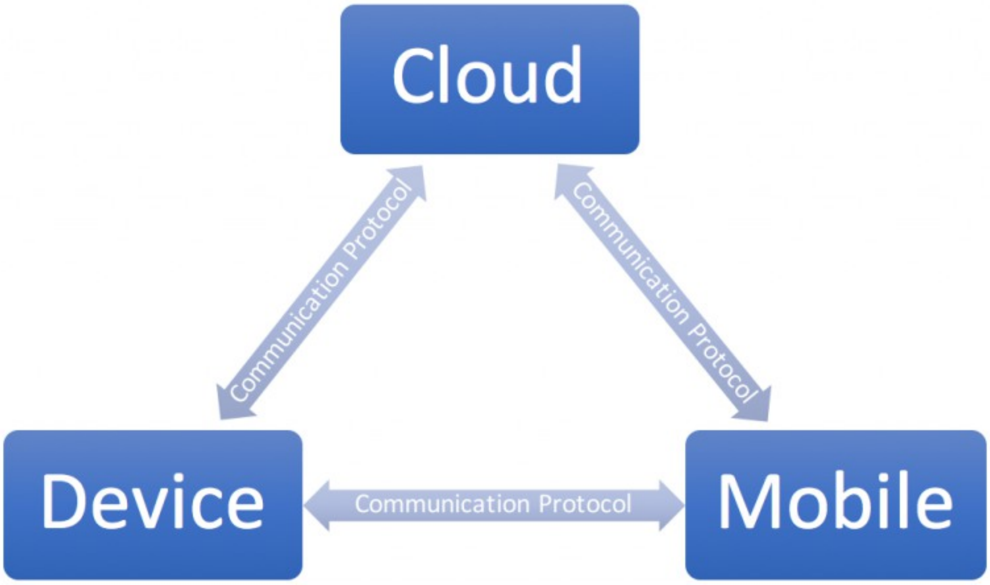
As we mentioned above, IoT is all about making us fat and lazy. Humans are good at innovating, and we can find out areas that can be improved event in a near-perfect system for whatever reasons. The usage of IoT technology is limitless in today's world. I bet if you just look around, you will probably come up with an IoT idea. There are various domains that are currently seeing a lot of innovation in IoT with the sole aim of automation and real-time data analysis from the physical world.

3. IoT Architecture

3.1 High level view

In its simplest form, IoT architecture includes three components as shown in the below diagram.

1. Mobile
2. Cloud
3. Device

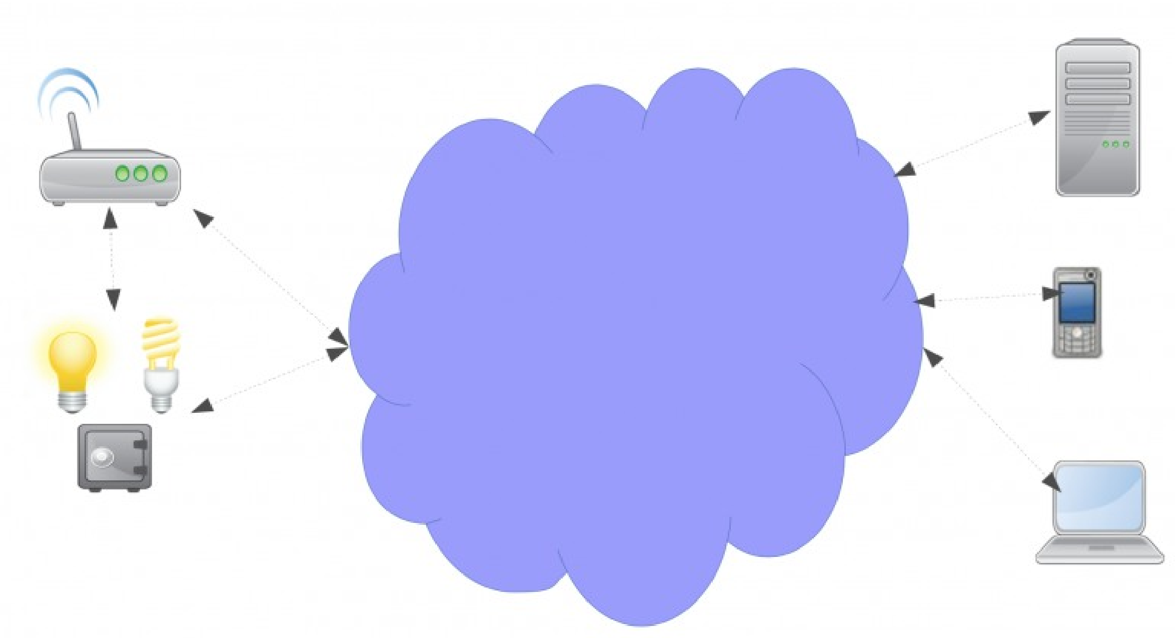


The communication between the components depends on the usage and/or type of the IoT product. Following are some examples that will make it clear how and why do components talk or do not talk to each other.

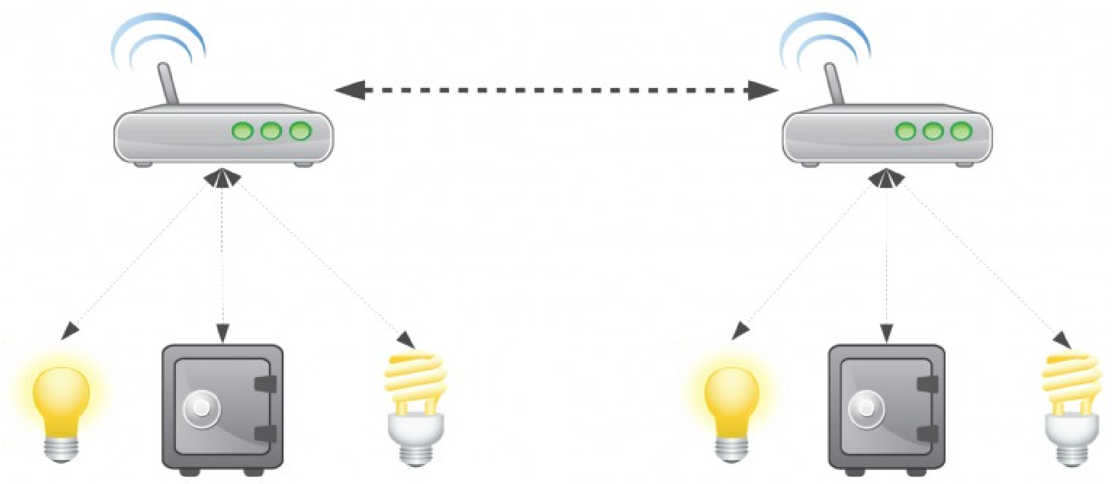
1. Device talks mobile-only – Ex. BLE based devices
2. Device talks to IoT gateway only – Ex. ZigBee, Wireless HART devices, etc.
3. Mobile talks to Cloud-only – In cases where the users do not have proximity access to the device, and it can only be controlled via the cloud.

3.2 Functional Architecture

Expanding further, the functional architecture can be defined as a network of sensors communicating with the cloud and the mobile/web interface via the Internet. The sensors may have their own network over traditional TCP/IP-based technology or Radio based in case where the traditional networks cannot be implemented, and radio communication offers more efficiency and makes more sense. In the latter case, there needs to be a gateway (our so-called IoT gateways/Hubs/Routers) that acts as an interface between the radio communication and the traditional TCP/IP communication. From now on, I will refer to TCP/IP as traditional network/communication.



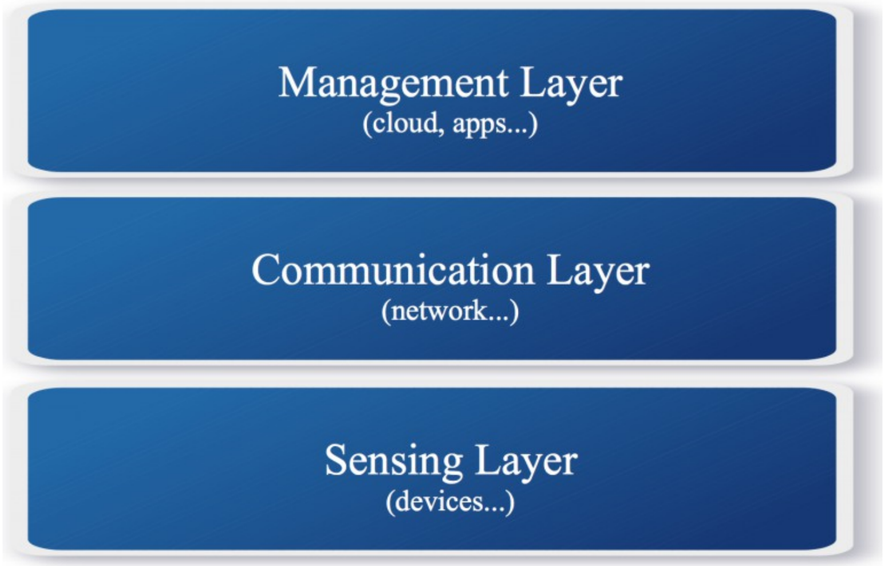
We can also have geographically spread sensor networks that can communicate with/are connected to each other via traditional networks by the IoT Gateways, as shown below.



3.3 Layered model

If we look at the IoT technology from a layered perspective, we can define it in 3 simple layers that form the core of IoT

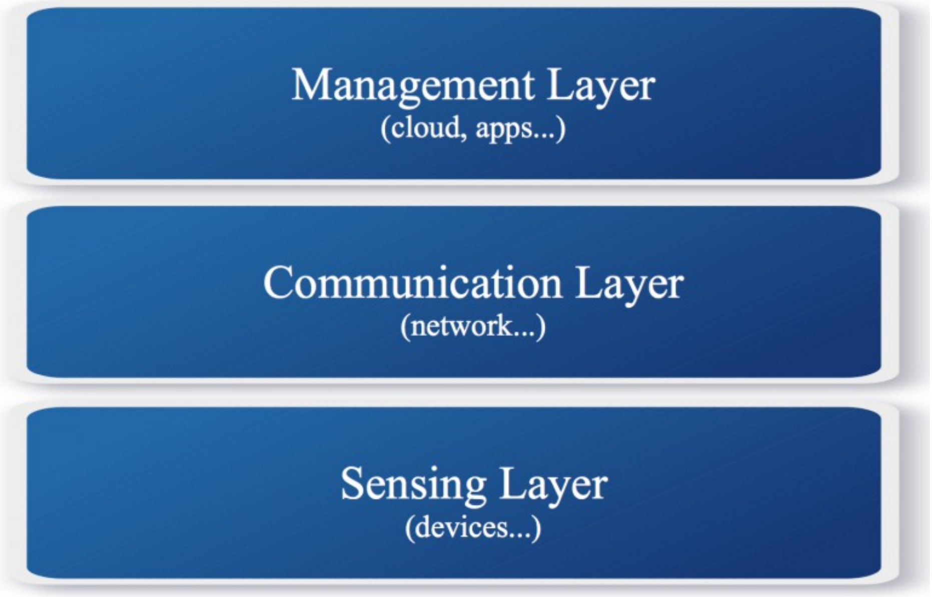
1. Sensing layer - This consists of the hardware sensors and sensor networks.
2. Communication Layer – This consists of the communication mechanism that allows the sensing layer to communicate with the Management layer, for example – WiFi, 3G, LTE, Ethernet, etc.
3. Management Layer – This is the top most layer and is responsible for making sense out of the raw data and provide a presentable and fancy view to the users. It includes the cloud, storage, apps, etc.



3.3 Layered model

If we look at the IoT technology from a layered perspective, we can define it in 3 simple layers that form the core of IoT

1. Sensing layer – This consists of the hardware sensors and sensor networks.
2. Communication Layer – This consists of the communication mechanism that allows the sensing layer to communicate with the Management layer, for example – WiFi, 3G, LTE, Ethernet, etc.
3. Management Layer – This is the top most layer and is responsible for making sense out of the raw data and provide a presentable and fancy view to the users. It includes the cloud, storage, apps, etc.



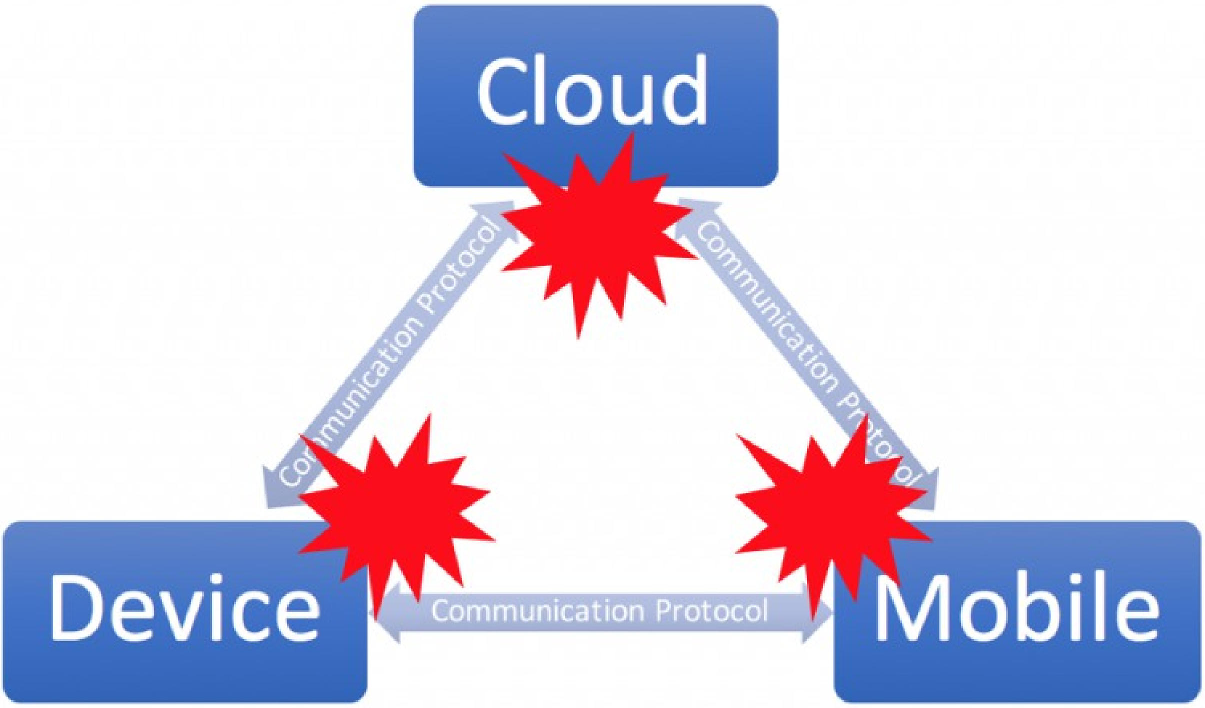
The aim of this chapter was to give you an idea about the IoT technology. Going forward, we will start talking about IoT security. The next chapter will describe the attack surface of the IoT ecosystem. I hope you enjoyed reading this as much as I enjoyed writing it.

Chapter 2: IoT Attack Surface

In this chapter, we will start getting into security and try to define a way to understand and create a structured process to perform security research or penetration testing of IoT.

If we look at the architecture defined in the previous chapter, it now becomes clear and easy for us to segregate the components of IoT and try to define the attack surface for each one of them individually and then combine them to create a holistic overview of the IoT ecosystem attack surface. I call it an IoT ecosystem instead of an IoT product because it indeed is an ecosystem of different components talking to each other and solving a particular real world problem. Let's go ahead and define the attack surface of the IoT ecosystem and discuss each component's attack surface in detail. The attack surface by components can be divided into three or four(if we include communication as an attack surface) major areas as follows:

1. Mobile
2. Cloud
3. Communication
4. Device



OWASP is also doing a lot of work in IoT security now. They have also defined the attack surface. I would again urge you to go through it. It is good to understand different ideas and thoughts as it helps you create your own comprehensive attack surface. OWASP is also doing a lot of work in IoT security now. They have also defined the attack surface. I would again urge you to go through it. It is good to understand different ideas and thoughts as it helps you create your own comprehensive attack surface.

Note:

1. The word microcontroller is used in a generic form to mean microcontrollers, microprocessors or SoC (System on a Chip) unless specifically mentioned with an explanation.
2. The below attack surface is defined by us and may be different from other sources.

1. Mobile

Mobile is one of the important user interfaces for IoT via which end users get insights into the state of the physical world. Since mobile app communicates with the IoT ecosystem to send commands and read data, it becomes one of the entry points into the IoT ecosystem. We will try to list down the attack surface for the mobile from an IoT perspective

1. Storage
2. Authentication
3. Encryption
4. Communication
5. Generic mobile vulnerabilities – OWASP Mobile Top 10 comes to mind

2. Cloud

The cloud is one of the very important pieces of IoT as usually, data from all the instances of the product line converges here. This makes it a very interesting attack point. Remember, I mentioned that IoT is not only about hardware in the previous chapter. The reason being that cloud will hold the data of all deployed IoT instances and has the privileges to send commands to all of them. Well, generally, it happens to be user-initiated, but if compromised, the attackers will gain control of the devices (and its data) deployed worldwide, which is dangerous. Overall the attack surface focuses on the interfaces that it provides, which include

1. Storage
2. Authentication
3. Encryption
4. Communication
5. Generic mobile vulnerabilities – OWASP Mobile Top 10 comes to mind
6. Generic Web/cloud vulnerabilities – OWASP Web Top 10 comes to mind

3. Device

Next is the device, which is the game-changer for IoT tech :). It interfaces with the physical world and also communicates with the virtual world. It is the first stop for physical world data. There is a whole debate around user privacy given the sensitive data about the user it stores (for example, home stats, body stats, personal information). In the future, devices may use user's cryptocurrencies directly through their wallet or a separate temporary wallet to purchase items, repairs, etc. The attack surface looks somewhat as follows

1. Storage
2. Authentication
3. Encryption
4. Communication

5. Sensor interface
6. Peripheral interfaces
7. Hardware interfaces
8. Human machine Interface

4. Communication

Although this is not a tangible attack surface as ideally, the tangible attack surface would be the communication interfaces and the respective drivers/firmware responsible for the communication. However, this needs a separate section on its own because there is an endless list of communication protocols that the IoT ecosystem can use on wired as well as a wireless medium. The following are some of the areas that make up the attack surface for the communication.

1. Authentication
2. Encryption
3. Deviation from the protocol standard
4. Protocol implementation anomalies

The hardware interfaces allow for actual communication. However, the actual data communication/packets are defined by the upper layers, which are implemented in the software. Hence, in this Attack surface area (communication), we will only discuss the protocols. Although the flaws in the protocol may result in attacks on the protocol endpoints residing on the mobile, device, or the cloud, we have kept it as a separate attack surface for clarity. There are way too many standards to mention in the list here. However, we will list down some of the common protocols that are used in various IoT products.

5. Web

The web, or in technical terms HTTP(S) is the most common protocol used for communication and is used everywhere. We dedicate a separate entry for this as the attack surface on the web is huge. However, the good news is that the attack surface, vulnerabilities, and mitigation techniques have mostly been standardized as it has been under research for more than two decades now. There are plenty of resources available online which describe the attacks and protection in detail. For starters, OWASP has done a great job with their Web Top 10, testing guide, and various open-source tools (www.owasp.org)

6. Others

Apart from web, there are many protocols, some domain-specific, some generic, and some for efficiency reasons. There are too many protocols to list down here. For brevity, we will list some of the common protocol standards to give you a fair idea about the kinds of protocols in use. History tells us that all protocols will have their share of implementation flaws, protocol design flaws, and configuration flaws. These need to be analyzed during a pentest.

1. CoAP - https://en.wikipedia.org/wiki/Constrained_Application_Protocol
2. MQTT - https://en.wikipedia.org/wiki/MQTT
3. AMQP - https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol
4. WebSocket - https://en.wikipedia.org/wiki/WebSocket
5. CANbus - https://en.wikipedia.org/wiki/CAN_bus
6. Modbus - https://en.wikipedia.org/wiki/Modbus
7. Profibus - https://en.wikipedia.org/wiki/Profibus
8. DNP3 - https://en.wikipedia.org/wiki/DNP3
9. BACNet - https://en.wikipedia.org/wiki/BACnet
10. HL7 - https://en.wikipedia.org/wiki/Health_Level_7
11. XMPP - https://en.wikipedia.org/wiki/XMPP
12. UPnP - https://en.wikipedia.org/wiki/Universal_Plug_and_Play
13. DNS
14. SSH
15. <The list goes on>

The above should give you a high-level overview of the attack surface for the IoT ecosystem. Now that we have a fair idea about it, let's define a detailed attack surface for the device so we know what exactly we need to attack in a standard IoT pentest. This is also helpful for IoT security architects to create a threat model for the IoT Product. Please note we are not going to (re) define the attack surface for Mobile and Cloud as you can find plenty of resources on the Internet describing the same. The idea of this e-book is to create a bridge for security researchers to get into IoT security so, we will focus on knowledge that is not currently available or structured. Given that we will still talk about mobile and cloud security wherever it is relevant for the IoT ecosystem from our perspective.

7. Device Attack Surface

Ok, let's do this! The following is a segregated and structured definition for the IoT attack surface. Please note that this is as per our understanding and has not been picked up from other sources.

7.1 Storage

The storage used by the device. This can further be segregated into Internal and external, Persistent and volatile.

7.1.1 SD Card

SD cards are typically used to store configuration and product data. They might be used to store firmware updates as well. It is a very interesting attack surface, and we will talk about certain attacks that are possible via the SD card in the upcoming chapters.

7.1.2 USB

Certain products may use USB drives to store similar data as in SD Cards as well as read data that is downloaded or stored on the USB drive. Similar attacks as for SD cards are applicable to USB storage.

7.1.3 Non-volatile Memory

These are used for various things, including read/write sensor data, bootloader, firmware, credentials, keys, etc. While testing the hardware board, it is crucial to look at the data stored on the chip. We can also perform a run-time analysis of the communication between the memory and the microcontroller to analyze what kind of data is stored/read during different operations. This is achieved by having a Logic analyzer sniff the bus communication. You can find interesting data being read/written while triggering specific operations on the device. There are different types of memory chips as follows:

1. EPROM
2. EEPROM
3. FLASH – More commonly used due to its speed and efficiency

An I2C Serial EEPROM

ST
24C02CB6
91B440

7.1.4 Volatile Memory

When talking about volatile memory, the word RAM immediately pops up in our mind. These are widely used in PCs and embedded systems and hold the code and data at run-time. The data is lost when the device is powered Off. Some of the common RAM types are as follows

1. SRAM (Static Random Access Memory) - A type of RAM that holds the data which is lost when the chip is powered off.
2. DRAM (Dynamic Random Access Memory) - Data is held for a period after which it is lost unless it is refreshed during run-time. This means that the data has a short lifespan even during the time the chip is powered on as compared to SRAM. The data is also lost when the chip is powered off.

7.1.5 Microcontroller Internal Memory

Microcontrollers also have their own internal memory, which is typically used to store code. These memories are usually accessible while debugging a microcontroller, for example, debugging via JTAG. The various memories that are used in microcontrollers are:

1. SRAM
2. EEPROM
3. FLASH

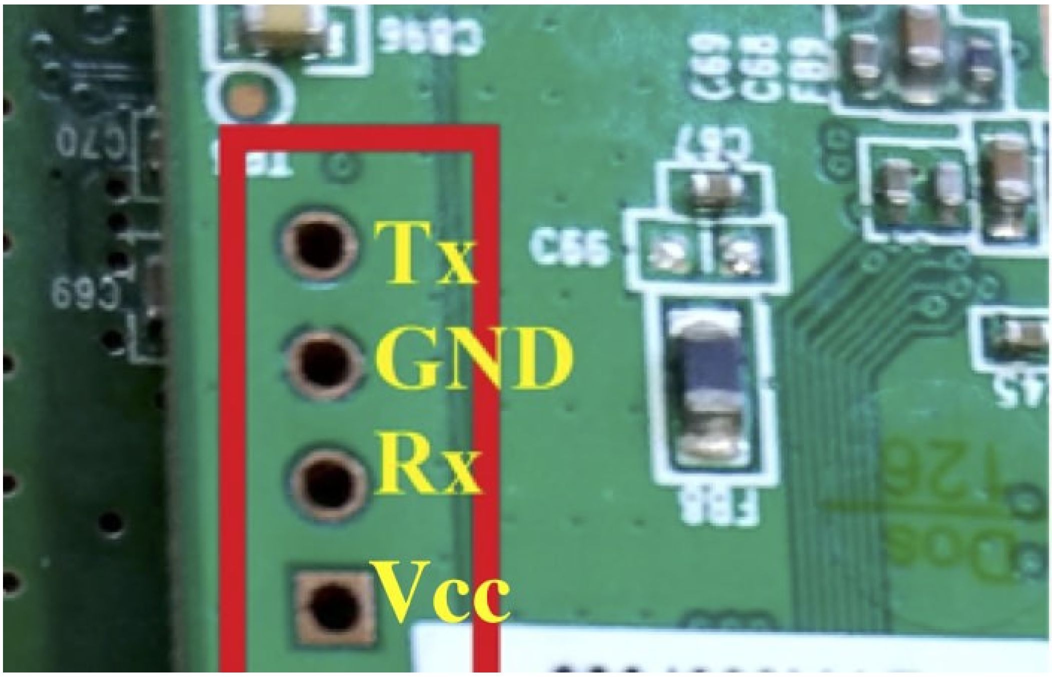
7.2 Hardware Communication Interface

Different hardware components on the same board need to talk to each other and to the outside world. All this communication is done using well defined and standard hardware communication protocols. From an attacker's perspective, it gives them an insight into the actual communication via sniffing or injecting malicious data. Some of the most common interfaces mentioned below should be analyzed for finding security issues.

7.2.1 UART

UART (Universal Asynchronous Receiver Transmitter) is a hardware component that allows asynchronous serial communication between two hardware peripherals. These can be on the same board (for example microcontroller talking to a motor or LED screen) or between two different devices (for example, a device microcontroller talking to a PC). It is an interesting attack surface as it may allow read/write access to the device over serial. In many devices, UART ports on the board are left open, which anyone can connect and access over serial to get a console ofsome sort, i.e., simple shell, custom command line consoles, log output, etc. A device will typically have a group of pin-outs connected to the microcontroller UART RX and TX pins, which are used for sending and receiving serial data. We will discuss in detail in the future chapters how to identify and access the UART ports on a device.

A typical 4-pinout UART port



7.2.2 Microcontroller Debug Port

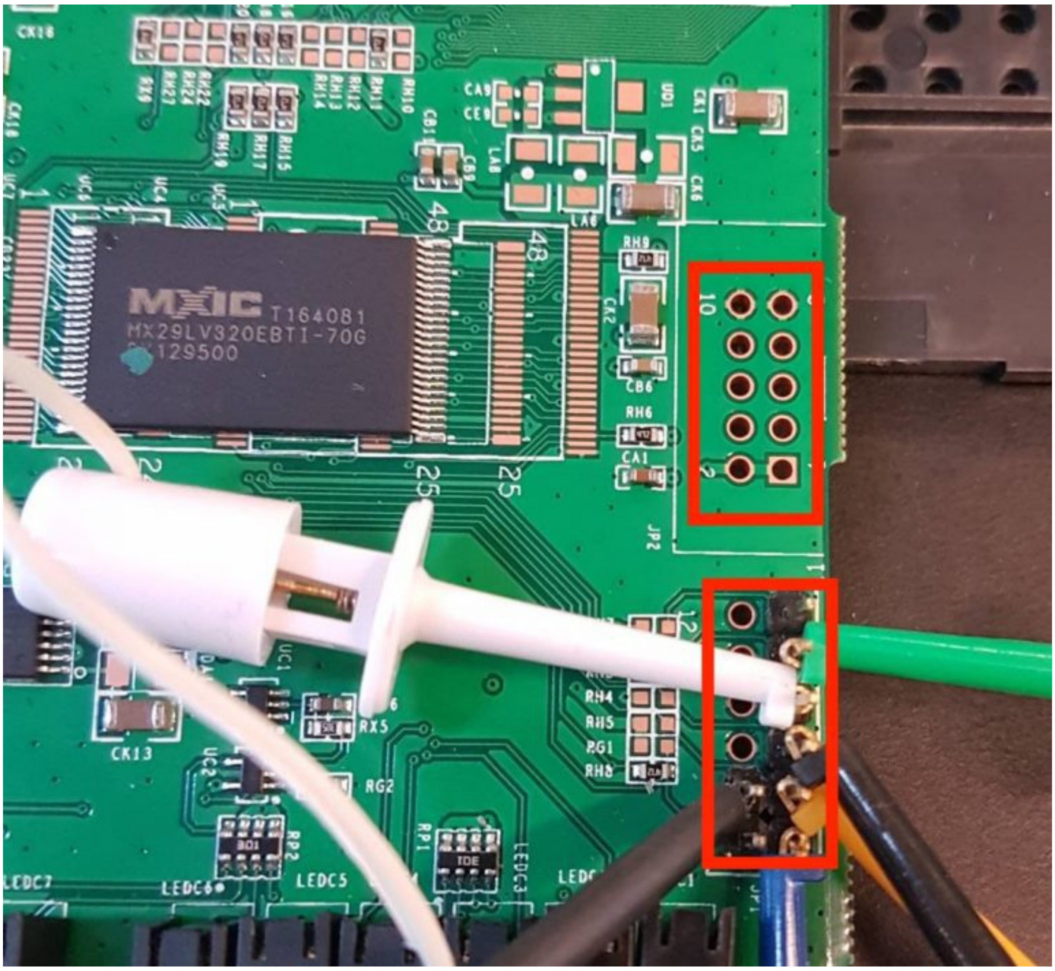
Microcontrollers have provisions for debugging during run-time using specified pins that are connected to pin-outs on the board. These pin-outs (ports) are used by the developers and designers to debug, read/write firmware and microcontroller internal memory, control/test microcontroller pins post-production. This makes the debug ports one of the most critical attack surfaces, given the power and access it gives to the attacker. There are a few standard interfaces used for this purpose, which are as follows:

1. JTAG (Joint Test Action Group): As the microcontrollers and PCBs were becoming smaller and smaller, it was getting difficult to test them after production. So, to efficiently test the boards post-production, the electronics Industry created an association with the same name and defined a method to test the Boards after production. It was later adapted as IEEE standard 1149.
    1. The JTAG protocol defines standard interfaces and commands that can be used to test and debug the microcontroller. JTAG defines four pin interface (and one additional optional pin TRST):
    2. TMS – Test Mode Select
    3. TCK – Test Clock
    4. TDI – Test Data In
    5. TDO – Test Data Out
    6. TRST – Test Reset (optional pin)

    In addition to testing the chips, these pins are used to by the debuggers to communicate with TAP (Test Access Port) which is implemented on the security perspective, Identifying the JTAG port and interfacing with it allows attackers to extract firmware, reverse engineer the logic, and flash malicious firmware on the device. More on it later, in the upcoming chapters.

2. cJTAG (Compact JTAG): This is a new JTAG protocol defined in the standard IEEE 1149.7. It does not replace 1149.1 standard but extends it further and is backwards compatible with JTAG. It defines a two-pin interface (TCK and TMS) and a new TAP that implements the new features.
3. SWD (Serial Wire Debug): SWD is another interface/protocol that is used for debugging microcontrollers. It is a two-pin interface:a. SWDIO (bidirectional)b. SWCLK (clock) It is an ARM specific protocols that uses ARM CPU standard bi-directional wire protocol, defined in the ARM Debug Interface v5. The benefit of SWD is that it claims to be more efficient than JTAG.

How a JTAG port might look like on a PCB board



Note that JTAG port may not necessarily be in a group of 10-pinouts as in the above image.

7.2.3 I2C

Inter-Integrated Circuit is a short-distance communication protocol used for communication between chips on the same board. It was invented by Philips (now NXP). It has a master-slave (multi) architecture and uses a two-wire bus

1. SDA - Serial Data
2. SCL - Serial Clock

One of the use cases of I2C is in EEPROM chips that are connected to the microcontroller I2C pins and typically store data or code. Typical attacks would include tampering with the data, extracting sensitive information, corrupting the data, etc. We should analyze the data at rest on the EEPROM chip as well as perform run-time analysis by sniffing the I2C communication to understand the behavior and security implications. As mentioned previously, we will have a dedicated chapter in the e-book for understanding and analyzing I2C communication.

7.2.4 SPI

Serial Peripheral Interface is also a short distance communication protocol used for communication between chips on the same board. It was developed by Motorola. It is full-duplex and uses master-slave architecture (single master). It also has higher throughput as compared to I2C. It uses a four-wire serial bus:

1. SCLK – Serial Clock. Other names include SCK
2. MOSI – Master Out Slave In. Other names include SIMO, SDI, DI, DIN, SI, MTSR.
3. MISO – Master In Slave Out. Other names include SOMI, SDO, DO, DOUT, SO, MRST.
4. SS – Slave Select. Other names include S S , SSEL, CS, C S , CE, nSS, /SS, SS#

It is used for talking to a variety of peripherals. Flash and EEPROM chips also use SPI. The methodology of testing and analyzing is similar to I2C, just that we have a different bus interface. We will talk about SPI in detail in the upcoming chapters.

7.2.5 USB

The device can have a USB (mini/micro etc.) interface for either charging or communication. For the latter, it becomes necessary to test the interface for known or unknown issues. We should sniff the communication for run-time analysis as well as fuzz the USB interface for unknown bugs.

7.3 Sensor

It is a loose name that we have given to mean the interface to the physical world. It may not necessarily be limited to a sensing type interface. For example, a temperature sensor would be a perfect example, but also a door lock that does not sense anything but controls the physical world by the “Lock/Unlock” action. These can be divided into three types based on their operation:

1. Monitor: This is more closely linked to the literal meaning of a sensor i.e. to sense

or monitor the physical world for any changes. Ex. temperature, motion, Pulse, Blood pressure, tyre pressure, etc.

2. Control: These types of devices control the physical world in some way or the other. Ex. Locks, dispensers, etc.
3. Hybrid: These are a combination of both the above types i.e. temperature control, Lights based on time of the day etc. This is one of the critical interfaces as all values and data derived from the physical world will be transferred to the cloud. If attackers can force the device with malformed (wrong) data then the whole ecosystem gets affected as all decisions and statistics are based on this data. In other words, this is the crux of the IoT ecosystem. Wrong values here can have catastrophic effects on the decisions that the ecosystem makes.

7.4 Human Machine Interface

As with the Sensor interface, we use the HMI as a general term to define the interface between the user and the device without restricting it to the term used in Industrial Control Systems. This is the interface that users can use to communicate with the device and operate on it directly. Some of the common examples would be touch screens, pushbuttons, touchpads, etc. It is important to test this interface to find out any bypass mechanisms, security flaws, etc.

7.5 Other Hardware interfaces

There are many other hardware interfaces used to communicate with the device. As a pentester,
it is important to analyze and find flaws and bypass mechanisms in all the interfaces. Some of the well-known interfaces include (but not limited to):

1. D-Subminiature – https://en.wikipedia.org/wiki/D-subminiature
2. Recommended Standards (RS232, RS485 etc) – More details on RS protocols can be found on https://en.wikipedia.org/wiki/EIA_standards
3. On-board Diagnostics (OBD) – https://en.wikipedia.org/wiki/On-board_diagnostics

8. Network Communication Interface

This interface allows the device to talk to the rest of the virtual world, which includes the sensor network, cloud, and mobile. The hardware interfaces responsible for network communication may have their own separate microcontroller/firmware that provides the communication functionality. The attack surface, in this case, is the firmware or driver code that implements the low-level communication.

8.1 Wi-Fi

The Wi-Fi interface has some known issues, etc. From the attack surface perspective, it would be interesting to attack the Wi-Fi chip, possibly to damage it, DOS, bypass security restrictions, or code execution.

8.2 Ethernet

The Ethernet interface, like Wi-Fi interface, has its share of low-level TCP/IP stack vulnerabilities as well as hardware implementation vulnerabilities and similar attack vectors.

8.3 Radio

The Radio interface has become one of the most important attack surfaces, considering many IoT products have shifted to/are being built with radio communication. The preference stems from the fact that it is more efficient to use Radio in many cases over Wi-Fi/Wired network connectivity. The reason I have categorized Wi-Fi separately and not in this section is primarily to make a clear distinction between devices that can connect directly to the internet (Wi-Fi/Wired) and the ones that require a gateway (ex. Smart Hubs) that implements both the Radio as well as Wi-Fi/Wired interfaces, to communicate with the sensors and the internet respectively. From the actual communication perspective, think of it as two different modes of communication:

1. Simple/Unstructured: This type is usually used in simple products like shutters, locks, doorbells, etc. By Simple and unstructured, we mean that it uses simple (mostly proprietary) data (stream) and sends it across via the radio interface. As a penetration tester, you need to reverse engineer the communication to find out flaws in the implementation. It is easy to sniff the radio communication using radio sniffing hardware tools like SDR (Software Defined Radio), etc.
2. Complex/Structured: Complex and structured communication means that it uses structured packets for radio communication, which are complex as they carry additional and metainformation about the protocol in addition to just the data. These protocols have become quite famous in the IoT world due to efficiency, standardization, economic chips, the convenience of implementation. Again, there are various tools available to sniff and parse the protocol to extract application specific data that is sent across. Some of the common protocols include:
    a. Bluetooth (and BLE)
    b. ZigBee
    c. Zwave
    d. NFC
    e. RFID
    f. LoRA
    g. Wireless HART

Chapter 3: IoT Top Ten Vulnerabilities

When talking about the Top Ten vulnerabilities, the first thing that comes to our mind is OWASP. Why not? After all, they are the pioneers in defining the top 10 vulnerabilities for web and mobile. We are OWASP fans simply because of the work the OWASP community has done over the years to define Application security issues, provide free tutorials and open-source tools for the Industry to mitigate the risks and vulnerabilities. It would be highly unlikely that you haven't heard of OWASP or read content from their website; however, if you have not, we would strongly suggest that you go through their website. https://www.owasp.org

OWASP has also started the IoT security initiative where the community has defined the IoT attack surface and the IoT Top 10 vulnerabilities in addition to web and mobile. They are in the right direction, and soon enough, it will be an excellent place for IoT security content. https://www.owasp.org

The content relevant to the reader for IoT security on the OWASP website is as follows:

1. OWASP Web Top 10 project:
   https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
2. OWASP Mobile Top 10 Project:
   https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
3. OWASP Internet of things project:
   https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project\
   a. OWASP IoT Attack Surface:
      https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT Attack_Surface_Areas\
   b. OWASP IoT Top Ten Vulnerabilities:
      https://www.owasp.org/index.php/Top_10_IoT_Vulnerabilities_(2014)

OWASP Internet Of Things Top Ten Vulnerabilities 2014

OWASP defined the first version of the Top 10 IoT vulnerabilities in 2014. They are quite comprehensive, and we would suggest that you go through them and understand what the threats and issues with the IoT ecosystem are. As homework, you can map it to the attack surface we defined in the previous chapter. The OWASP IoT top ten vulnerabilities (as per https://www.owasp.org/index.php/Top_IoT_Vulnerabilities) are as follows:

I1. Insecure Web Interface
I2. Insufficient Authentication/Authorization
I3. Insecure Network Services
I4. Lack of Transport Encryption/Integrity Verification
I5. Privacy Concerns
I6. Insecure Cloud Interface
I7. Insecure Mobile Interface
I8. Insufficient Security Configurability
I9. Insecure Software/Firmware
I10. Poor Physical Security

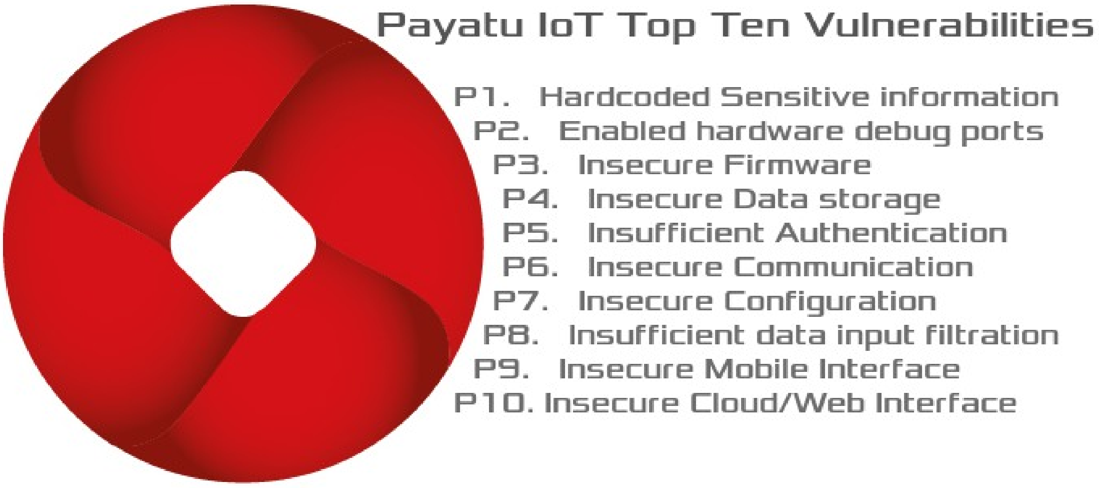
We will not go into the details of each item in the top ten list. The details can be found on the OWASP website. Instead, We refined the top ten based on our experience of issues found by us or issues published on the Internet.

Payatu Internet of Things Top Ten Vulnerabilities 2018

Disclaimer: Please note, our aim was not to try to override OWASP Top Ten, the guys have done a great job. Kudos to the OWASP team! This was rather an exercise based on our experiences and focusing more towards the hardware and new IoT technology which deserves its due attention.

We will continue to maintain and update Payatu IoT Top 10 Vulnerabilities. We combined web and cloud into one item, the reason being that not all sensors or IoT devices will have a web interface and cloud is an important part of the ecosystem, which is mostly web API based from an attack surface perspective. Also, some of the vulnerabilities may be applicable to more than one component. For example, Hardcoding is applicable to both devices and mobile app.

After a lot of research and analysis on the common issues across IoT, we created the Payatu Top 10 IoT vulnerabilities that have caused an impact on the IoT security market and product. We will explain all below IoT vulnerabilities to provide an understanding of the basic security issues.


Payatu IoT Top Ten Vulnerabilities
P1. Hardcoded Sensitive information
P2. Enabled hardware debug ports
P3. Insecure Firmware
P4. Insecure Data storage
P5. Insufficient Authentication
P6. Insecure Communication
P7. Insecure Configuration
P8. Insufficient data input filtration
P9. Insecure Mobile Interface
P10. Insecure Cloud/Web Interface

P1. Hardcoded Sensitive information
P2. Enabled hardware debug ports
P3. Insecure Firmware
P4. Insecure Data storage
P5. Insufficient Authentication
P6. Insecure Communication
P7. Insecure Configuration
P8. Insufficient data input filtration
P9. Insecure Mobile Interface
P10. Insecure Cloud/Web Interface

P1. Hardcoded Sensitive information

Hardcoding information during development is common practice as developers hardcode static data within a program. However, the problem occurs when sensitive information is hardcoded. It is very likely to have sensitive information hardcoded within the firmware as well as the mobile App or a thick client. The issue is that it remains the same for all the instances of the product and can be used for attacking any product instance deployed in the field. Some examples of sensitive information that is hardcoded:

1. Credentials – of device services, cloud services, etc.
2. Encryption keys – Private keys, symmetric encryption keys
3. Certificates – client certificates, etc.
4. API keys – Private/paid APIs
5. URLs – development, firmware related, user related, backend, etc.
6. Configuration

P2. Enabled hardware debug ports

The device hardware may have debug ports open for interaction with the system. In simple terms, it is a set of pins-outs on the PCB which are connected to the microcontroller / microprocessor pins, and you can use a client software to connect to these pin-outs to communicate over a hardware communication protocol that allows you to interact with the system. The level of interaction and privilege is dependent on the type of protocol and its usage. There may be pinouts for the UART interface, for example, which may give you access to high-level software/application, i.e., command shell, logger output, etc. You can also get a low-level interaction with the micro-controller using protocols such as JTAG, SWD, etc. These give you direct control over the micro-controller so you can test and analyze the microcontroller pin values, read/write the internal flash, read/write register values, debug the OS/base firmware code, and much more. If these ports/pin-outs are enabled on the device, an attacker can hijack the device and/or extract sensitive information from the device, including the firmware and data. These ports are usually enabled for troubleshooting/debugging issues in production devices.

P3. Insecure Firmware

The term “insecure” here refers to the way firmware is managed and not specifically firmware code vulnerabilities themselves. Firmware contains the business logic of the device and is mostly proprietary, i.e., IP (intellectual property) of the vendor. If an attacker has access to the plaintext firmware, he/she can reverse engineer it to find security issues or to clone the logic and ultimately the product itself. The vulnerabilities depend on the way firmware is stored and updated on the device. If care is not taken to properly encrypt the firmware in storage or in motion (updates), the attackers can get hold of it. Some of the issues with firmware are (but not limited to):

1. Firmware is stored in plaintext on memory chips
2. Firmware is not signed and/or bootloader does not verify the integrity of the firmware before loading
3. Firmware updates are transported in plaintext from the cloud or mobile to the device.
4. Firmware updates are transported over a plaintext communication protocol, for example, HTTP.
5. Firmware encrypted with a single symmetric key for all the device instances.
6. Firmware encryption keys transferred along with the update to the device.

A properly implemented PKI based system can ensure optimum security. However, most low power sensors lack the computation power to implement PKI effectively. Also, if updates are secure but the key can be extracted from the device using other vulnerabilities, then the whole exercise is futile.

P4. Insecure Data storage

This issue is prominent in devices as well as mobile apps. It is more apparent in device hardware, probably due to the assumption that reversing hardware is difficult. Sensitive data, if not stored securely, can be extracted and utilized by an attacker to subvert the system. In addition to security issues, it may also have privacy implications if users' personal data is not protected properly. Some of the common issues:

1. Sensitive data stored in plaintext on memory chips
2. Sensitive data stored encrypted, but, encryption key is accessible
3. Custom encryption used to encrypt data
4. No access control for modifying data
5. Insecure data storage on mobile app (refer "P9. Insecure Mobile Interface")

P5. Insufficient Authentication

The devices may use improper or no authentication mechanisms, which allow the attacker to bypass the authentication mechanism altogether if it is implemented poorly and send unauthorized commands to the device. This is a serious problem for critical IoT devices as anyone on the network (TCP/IP or radio) can override the normal operations and control the device. Some of the authentication issues that occur on the devices are (but not limited to):

1. No client authentication
2. Authentication over plaintext communication channel
3. Improper encryption used for credentials
4. Predictable credentials
5. Default credentials

P6. Insecure Communication

The communication within the IoT ecosystem may be insecure if attackers are able to sniff, analyze, replay and extract sensitive information from the communication. The vulnerabilities may be due to using insecure communication protocols or protocol deficiencies themselves. To keep things simple, vendors may choose to use insecure methods of communication. Since IoT is a new and evolving technology, many IoT protocols do not define proper security mechanisms, or vendors implement default insecure modes. The issues include (but not limited to):

1. Unencrypted communication while sharing sensitive information
2. Using custom encryption
3. Using custom/proprietary protocols
4. Improper encryption used
5. Using protocol default (weak) security mode
6. Using protocols with known issues
7. Replay issues

P7. Insecure Configuration

This issue occurs when the device configuration is insecure or if the device does not allow users to modify configuration parameters. This issue also occurs in mobile app and cloud configuration. To keep things simple or shipping the product fast, developers may opt to use simple but insecure configuration and/or disallow changes. Some apparent issues are (but not limited to):

1. Using default insecure configuration
2. Disallowing integrators and/or users from modifying the configuration
3. Insecure low-level protocol and hardware configuration in release products
4. Insecure encryption modes and settings
5. Low or no visibility on user's personal data shared or stored

P8. Insufficient data input filtration

This is going to be a major issue going forward as more IoT protocols are implemented in the IoT ecosystem. The telemetry data coming from the device, for example, maybe trusted by the cloud or an IoT gateway, leading to known and unknown security issues such as remote code execution, web-based attacks like SQL injection, cross-site scripting, and many more. We expect this to move up in priority in the future. While mature implementations do filter data for traditional technologies, the same is yet to pick up for new IoT protocol implementations.

P9. Insecure Mobile Interface

As mobile technology is mature compared to sensor technology from a security perspective, we have grouped all mobile security issues into one. This does not mean they have less priority. As you can see, some of the high priority vulnerabilities are applicable to mobile as well. However, owing to the maturity of the technology, it already has a plethora of information on security issues and secure implementations. Being a fan of OWASP, we recommend starting from OWASP Mobile Top Ten vulnerabilities that will take care of the majority of the security issues.

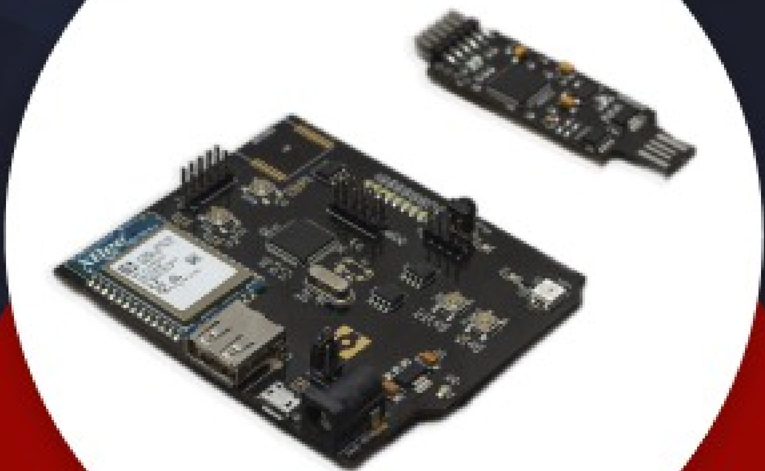
P10. Insecure Cloud/Web Interface

As discussed in "P9. Insecure Mobile Interface", the same applies to cloud and web. In case a device has a web interface, you may still be able to own the device via web attacks. However, these security issues are already well defined and understood. Again, we would recommend starting from OWASP Web Top Ten vulnerabilities for understanding and mitigating web security issues and documents from the Cloud security alliance for cloud security. Please note, this is not the only knowledge base available, and one should look at tools and research papers available on the internet for the same. It is important to note that the cloud forms the data storage and communication backbone for the IoT ecosystem. If the cloud is compromised, it may lead to the compromise of the whole IoT ecosystem, including all deployed products around the world and the Universe.

Reference :

Chapter 2 : IoT Attack Surface

- UART - https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
- Types of memory - https://en.wikipedia.org/wiki/Semiconductor_memory
- JTAG – https://en.wikipedia.org/wiki/JTAG
- SWD – https://www.arm.com/files/pdf/Serial_Wire_Debug.pdf
- JTAG Vs. SWD – https://electronics.stackexchange.com/questions/53571/jtag-vs-swd-debugging
- I2C – https://en.wikipedia.org/wiki/I%C2%B2C
- IoT Protocols – https://www.postscapes.com/internet-of-things-protocols/
- I2C Serial EEPROM image source – https://upload.wikimedia.org/wikipedia/
- commons/thumb/7/71/AT24C02_EEPROM_1480355_6_7_HDR_Enhancer.jpg/180px-AT24C02_EEPROM_1480355_6_7_HDR_Enhancer.jpg
- JTAG port image source – https://i.stack.imgur.com/IiUqm.jpg



Part 2

PROTOCOLS

Chapters: 4 to 9

What to expect:

Educate yourself with the essentials as we go over a general synopsis of various protocols within the IoT ecosystem including MQTT, CoAP, BLE, and ZigBee.

MQTT
Chapter 4: Introduction To MQTT Protocol And Security

In this chapter, we are going to look at one of the most famous and widely used IoT protocols - MQTT, security issues and attacks on MQTT. But before that, let's first get an idea of what exactly is MQTT and why is it the apple of IoT vendors' eye. Please note that we will use the terms Client and Node as well as Server and Broker interchangeably to mean the same thing.

1. Introduction

MQTT is short for Message Queueing Telemetry Transport. It is an OASIS and ISO Standard. It is based on the principle of the client/Server publish/subscribe message mechanism. As with other IoT protocols, it is lightweight and simple to use and implement, which works well for constrained and automated environments like M2M (Machine-to-machine) and IoT. It is an application layer protocol that runs over TCP and can run over any other network protocol that provides reliable, ordered, and bi-directional communication (e.g., WebSockets).

TCP Port	Communication Type
1883	Plain text
8883	TLS

The latest version of the protocol at the time of writing this is 5.0. The protocol specification can be found here -
https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

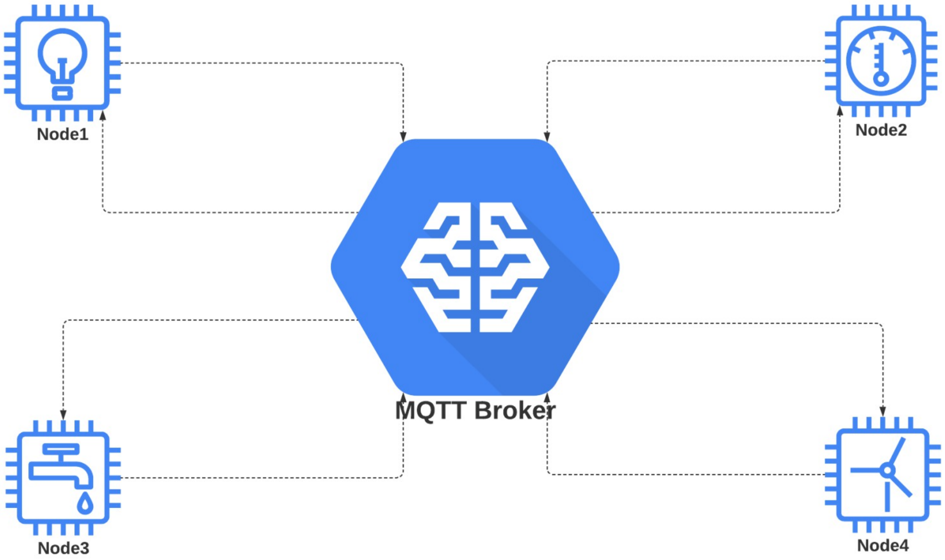
1.1 Use and Popularity

MQTT has gained a lot of popularity in the Industrial Internet of Things (IIoT) eco-systems, Industrial Control Systems (ICS), Enterprise IoT, etc., given its simplicity, flexible architecture, and cloud readiness. We think MQTT is going to play an extremely vital role in Industrial Control Systems (ICS) and IIoT infrastructures in the near future.

1.2 Architecture Overview

At the center of the MQTT, the network is an MQTT broker. All the MQTT nodes connect to the broker. Think of it as a star network. A node can subscribe for as well as publish messages. The nodes communicate with each other via the broker by publishing or subscribing to the broker. In simple terms, the broker forwards the messages from one node to the others based on subscriptions.).

Fig 1. A simplified view of the MQTT network



What components of the IoT eco-system will act as a broker and the nodes are totally dependent on the requirements of the Infrastructure, eco-system, and implementation.

A node can be (but not limited to):

1. An IoT edge device
2. An IoT Gateway
3. An application Server
4. Mobile Application
5. Cloud Application
6. Another MQTT broker (when bridging)

Similarly, a broker can be (but not limited to):

1. An IoT Gateway
2. A Cloud-based MQTT server
3. Mobile Application

2. MQTT Communication

2.1 Communication Overview

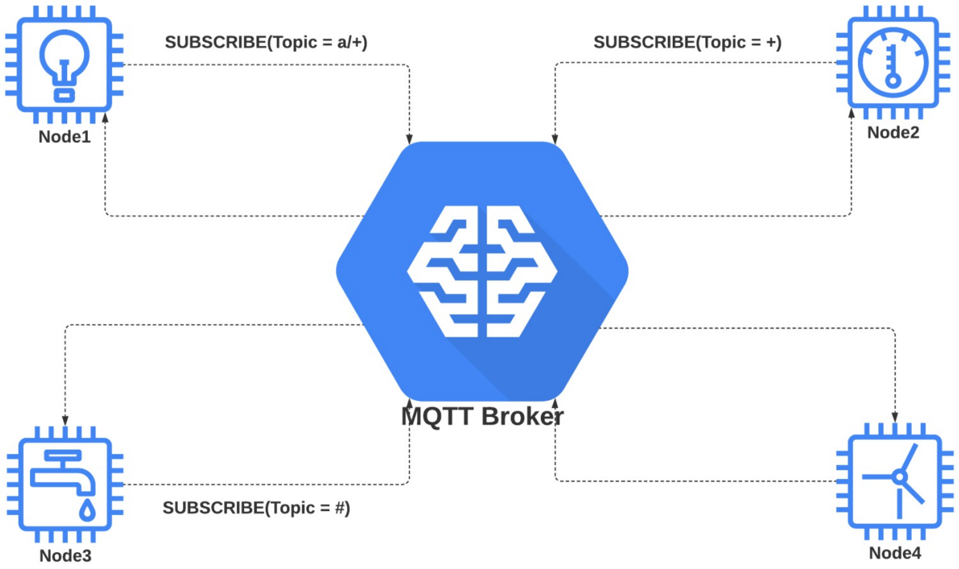
So, how do nodes specify whom to send messages to and to accept messages from? That is done using something called Topic. The nodes basically publish a message on a topic and other nodes can specify to the broker which topics they want to subscribe to. It's probably time to define some terminology and explanation about topics

1. `Topic` - A topic or topic name is just a label that is attached to a Message (Application payload). The syntax of a topic name is similar to a tree structure (Think file system tree, without a starting /). A valid topic for example could be payatu/office1/boardroom/temperature.
2. `Topic Level` - Each item in a topic represents a topic level and the topic levels have a parent-child relationship, for e.g. in payatu/office1/boardroom/temperature office1 is at a parent topic level of boardroom.
3. `Topic level separator` - Each topic level is separated by a forward slash / .
4. `Topic Filter` - A node can subscribe to one or more topics in a single request to an MQTT broker by specifying a topic filter. A topic filter is basically an expression that matches one or more topics (think simple regex). There are two wildcards that can be used to match multiple topics.
    1. Single-level wildcard `+` - is used to match any item in a specific topic level. Let's look at some examples to make it clear:
        1. `a/+/c/d` topic filter will match `a/b/c/d` , `a/x/c/d` but not `a/b/y/d` .
        2. `a/b/c/+` topic filter will match `a/b/c/d` and `a/b/c/x` but not `a/x/c/d` or `a/b/c/x/y`
    2. Multi-level wildcard `#` - It is used to match any item at the specific topic level and all its child topic levels. It must be specified at the leaf of the topic name. Some examples for `#` wildcard:
        1. `a/b/#` topic filter will match any topic level under "b" and any of its children i.e. `a/b/c` , `a/b/c/d` . It will not match `a/x/c` or `a/b`
    3. Both the wildcards can be used in a single topic filter e.g. `a/+/c/#`
    4. Single wildcard character filters are also valid for example # and +
    5. `Topic starting with $ character` - The topic names starting with the $ character (for e.g. $foo/bar/boo) are generally used for MQTT server implementation and management purposes. `$SYS/` is a widely-used topic prefix for server-specific information or control APIs as per the spec. However, this is not a mandate. Server implementations are free to use any prefix of their choice for e.g. AWS IoT uses the topic prefix $aws/ for all internal/management stuff. There are certain conditions and restrictions applied to these topics:
        1. The MQTT server will not match the topic filters starting with wildcard `#` or `+` with topic names starting with $ character.
        2. The nodes should not use topic names starting with $ character for exchanging messages with other nodes. In other words, they should not be used for application specific messages.

2.2 Communication Example

The below example demonstrates a scenario where we have 4 nodes connected to an MQTT broker. Three nodes(Node1, Node2, and Node3) subscribe to topics of their choices using different topic filters. One node (Node1) then publishes a message on a topic to the broker. Out of the three subscribing nodes, the broker publishes the message to only two nodes (Node2, and Node3) as their topic filters match the topic on which the message is published.

Fig 2. Three nodes subscribe to different topic filters



SUBSCRIBE(Topic = a/+)
SUBSCRIBE(Topic = +)
Node1
Node2
MQTT Broker
SUBSCRIBE(Topic = #)
Node3
Node4

Image Source Link : https://payatu.com/blog/aseem/iot-security---part-10-introduction-to-mqtt-protocol-and-security

Fig 2. Three nodes subscribe to different topic filters

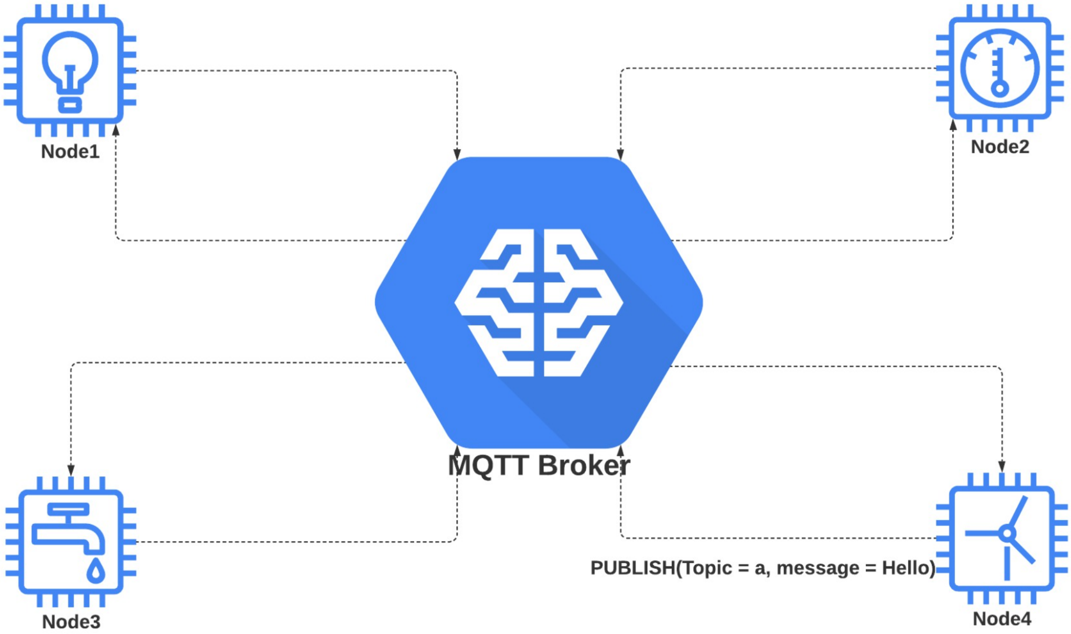
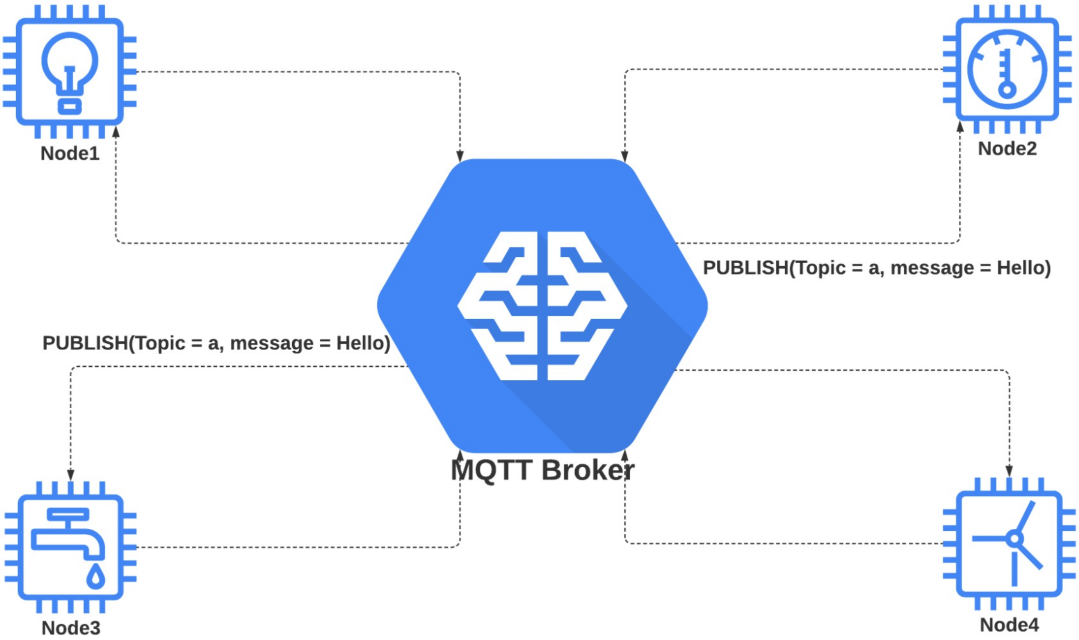

Node1
Node2
MQTT Broker
Node3
PUBLISH(Topic = a, message = Hello)
Node4

Fig 4. Two nodes receive the message "Hello" on the topic "a" because their subscription topic filters match the topic "a"


Node1
Node2
PUBLISH(Topic = a, message = Hello)
PUBLISH(Topic = a, message = Hello)
MQTT Broker
Node3
Node4

Assuming the same setup as above, If Node1 published messages on the below-mentioned topics to the broker, the nodes that will receive the messages from the broker for each topic are -

Topic on which message is published	**Nodes That will receive the Message**
a/b	Node1 and Node3
a/b/c	Node3
$SYS/broker/foo	None

3. MQTT Protocol Intro

Since this chapter focuses on the introduction, we will not go into the protocol's internal details. However, we will discuss the important aspects of the protocol and communication here.

3.1 MQTT Control Packets Intro

The communication between the client (nodes) and the server (broker) happens over MQTT Control Packets. An MQTT control packet consists of one to three sections:

1. 1. Fixed Header - This is present in all packets. This specifies the control packet type, flags, and the remaining packet length.
2. Variable Header - This is present in some packets. The contents of this header vary depending on the packet type. Usually, Packet Identifier is one of the fields in some packet types.
3. Payload - This is present in some packets like CONNECT, PUBLISH, SUBSCRIBE, SUBACK, UNSUBSCRIBE, and UNSUBACK. In the PUBLISH packet, the payload is the application message and is surprisingly optional. There are 16 MQTT control Packets defined in the MQTT version 5.0 specification.

Packet Type	**Description**
Reserved	Reserved
CONNECT	Connection Request
CONNACK	Connection Acknowledgement
PUBLISH	Publish a message
PUBACK	Publish Acknowledgement (QoS = 1)
PUBREC	Publish Received (QoS = 2)
PUBREL	Publish Release (QoS = 2)
PUBCOMP	Publish Complete (QoS = 2)
SUBSCRIBE	Subscribe Request
SUBACK	Subscribe Acknowledgement
UNSUBSCRIBE	Unsubscribe Request
UNSUBACK	Unsubscribe Acknowledgement
PINGREQ	Ping Request
PINGRESP	Ping response
DISCONNECT	Disconnect notification
AUTH	Authentication exchange

3.2 Quality of Service (QoS)

The protocol defines three levels of communication reliability via the Quality of Service flag. The value of this flag determines how the communication between the sender and the receiver will take place i.e. using different control packets. We mentioned the terms sender and receiver here because both client (node) and server (broker) can act as either sender or receiver depending on who is sending the message to whom, for e.g. it could be from node to broker or from broker to node. Let's look at the three levels:

1. QoS level 0 - At most once delivery - This is an unreliable messaging mechanism where no acknowledgment is sent by the receiver which means if the message is lost, it's lost.

Sender	Flow	Receiver
Sends PUBLISH with QoS = 0. Transaction complete	---->	Receives the message (or not)

2. QoS level 1 - At least once delivery - This ensures that the message is received at least once. The receiver is supposed to send the PUBACK packet in response to a P UBLISH packet.

Sender	Flow	Receiver
Sends PUBLISH with QoS = 1	---->	Receives the message
Receives the PUBACK. Discards message. Transaction complete.	<----	Sends PUBACK

3. QoS level 2 - Exactly once delivery - This is used when loss or duplication of the message is not acceptable. It is a 4-step process.

Sender	Flow	Receiver
Sends PUBLISH with QoS = 1	---->	Stores the Packet ID. May Initiate onward delivery.
Discards message. Stores PUBREC with Packet ID complete.	<----	Sends PUBREC with the same Packet ID
Sends PUBREL with same Packet ID	<----	Discards Packet ID for any future packets received ID
Discards stored state. Transaction complete	<----	Sends PUBCOMP with the same Packet ID

3.3 Packets and Usage

We will discuss some of the actions (and respective packets, their contents) briefly as describing all control packets along with each field would be an overkill for this 101-style e-book. However, we would urge you to go through the protocol specification to get a good understanding of the internals.

3.3.1 CONNECT

This is a logical MQTT connection request. This is the first MQTT control packet that is sent from the client to the server after the underlying connection is established. It contains some important information for the connection. Some of the important info in the variable header of the packet includes:

1. `Protocol` - It includes protocol name and version
2. `Connect Flags` - One byte specifying the corresponding values being present in the payload or not, things like user name, password, Will flag, Clean start, etc. The 'Will' stuff is basically used by the client to tell the server to send a specific (will) message on the will topic and utilize other will data if the client disconnects abruptly.
3. `Keep Alive` - Time in seconds between two-packet transmission. If elapsed, the client must send a PINGREQ packet.
4. `Properties` - Some packets, including CONNECT, contain standard properties, defined in the spec, at the end of the variable header.

Some of the important information in the payload includes:

1. `Client Identifier` - Each client connecting to the server specifies a unique client ID. This is used for maintaining the session state. Note that this is not part of the MQTT authentication.
2. `User Name` - The payload contains the user name if the user name flag is set in the variable header.
3. `Password` - The actual password, if the password flag is set in the variable header.

3.3.2 CONNACK

We will discuss some of the actions (and respective packets, their contents) briefly as describing all control packets along with each field would be an overkill for this 101-style e-book. However, we would urge you to go through the protocol specification to get a good understanding of the internals.

3.3.3 PUBLISH

This is used to publish an Application Message on a topic. The variable header includes:

1. `Topic Name` - The topic to publish the message on.
2. `Packet Identifier` - Present only for packets with QoS = 1 or 2 as it is required to send therespective acknowledgment packets.
3. `Properties` - Publish specific properties. Check the specification for more details

The payload of the packet contains the Application Message being published. The format of the data in the payload is application-specific. So, developers are free to decide what kind of data they want to exchange, such as JSON, XML, text, binary, etc. For e.g., AWS IoT uses JSON format for data exchange between AWS things and IoT core.

The response to PUBLISH depends on the QoS.

1. `QoS = 0` - No response
2. `QoS = 1` - PUBACK
3. `QoS = 2` - PUBREC (followed by an exchange of PUBREL and PUBCOMP)

3.3.4 SUBSCRIBE

It is used to subscribe to one or more topics for receiving Application Messages being passed on the same. The variable header contains a Packet Identifier and Properties. The payload contains a list of Topic filters, and each topic filter is followed by a byte describing its subscription options.

3.3.5 SUBACK

This packet is the response for the SUBSCRIBE packet and contains information about the receipt and the processing of the subscribe request. It can selectively grant or reject subscription requests for topic filters based on any issues. The variable header contains the packet identifier and Properties. The payload contains a list of Subscribe Reason Codes where each reason code corresponds to the respective Topic filter in the subscribe request i.e., the order of reason code is the same as the order of topic filters in the subscribe request. A reason code is a single byte that specifies success, failure, errors corresponding to the respective topic filter such as 'Not Authorized', 'Topic Filter Invalid', 'Granted QoS 0', 'Wildcard subscriptions not supported' to name a few. There are 12 Subscribe Reason Codes defined in the specification.

The other packets include:

1. PINGREQ and PINGRESP are used for connection keep-alive or heartbeats
2. AUTH is used when an extended or external authentication mechanism is supposed to be used in the MQTT session
3. DISCONNECT to close the MQTT session.

3.3.6 UNSUBSCRIBE

This packet is used to unsubscribe from one or more topics. The variable header contains the packet identifier and Properties. The payload contains a list of topic filters that the client wants to unsubscribe from.

3.3.7 UNSUBACK

This packet is a response to an UNSUBSCRIBE packet to confirm the receipt of the unsubscribe request. The variable header contains the packet identifier and Properties. The payload contains a list of Unsubscribe Reason Codes where each reason code corresponds to the respective Topic filter in the unsubscribe request i.e. the order of reason code is the same as the order of topic filters in the unsubscribe request. A reason code is a single byte that specifies success, failure, errors corresponding to the respective topic filter such as 'Success', 'Not Authorized', 'Topic Filter Invalid' to name a few. There are 7 Unsubscribe Reason Codes defined in the specification.

The other packets include:

1. PINGREQ and PINGRESP are used for connection keep-alive or heartbeats
2. AUTH is used when an extended or external authentication mechanism is supposed to be used in the MQTT session.
3. DISCONNECT to close the MQTT session.

4. MQTT Security

Now that we understand how MQTT works, let's focus on the security aspect of MQTT both from an offensive as well as defensive perspective. All the information below is based on our experience with MQTT security research and IoT product and infrastructure penetration testing projects. Below we will try to answer some of the questions that generally come up when one encounters a new protocol in their IoT penetration tests.

4.1 Tools

Are there any tools available that assist in MQTT analysis and assessment? Thankfully there are some choices available:

1. Open-source libraries, clients, server implementations. The only issue here is that they are rigid and have no guidance from the security perspective and you may end up editing or writing your own scripts to do the job.
2. EXPLIoT Framework has some pretty neat MQTT plugins for publish, subscribe and even MQTT Authentication cracking. It can interact with standard MQTT servers as well as AWS IoT MQTT servers.
3. Fuzzing - Scapy has MQTT packet creation capabilities and this can be used to create your own fuzzers.

4.2 Recon

Ok, so we have the tools, now what? Where do we start? The first thing is to perform recon on the MQTT broker to get as much information about the MQTT network as possible.

1. Can you connect without authentication?
2. What happens when you subscribe to all topics i.e. `#`
3. Do you have access to any of the `$SYS/` topics? What is the information available?
4. Are you able to read any sensitive information?
5. Are you able to publish to any or sensitive topics?
6. What topics are important and used in the IoT Eco-system? This can also be found by reverse-engineering the device firmware or Mobile App (if it is part of the MQTTnetwork).
7. How are Client ID generated? Is there any pattern to the client IDs?
8. Is TLS used?
9. How is the telemetry data sent from the device utilized on the cloud?

4.3 Analysis

Once we gather the basic information about the MQTT implementation, we can utilize it to launch different attacks on the eco-system. By eco-system, I mean all the components of the MQTT infrastructure, including the devices, gateways, Cloud Broker and apps, mobile apps, etc. We will list down some of the ideas to get your grey cells working:

1. Subscribing to `#` helps you understand which topics you have access to with or without authentication as well as the messages that are passed.
2. If the Broker requires authentication, you can launch password brute-force or dictionarybased attacks
3. Reversing the firmware or the mobile app may give you sensitive information such as
    1. Client Ids
    2. Credentials
    3. Certificates and keys
    4. Topics used
    5. What data is exchanged, and expected on specific topics and how it affects the behavior of the components.

4.4 Attacks

4.4.1 Denial of Service via Duplicate Client ID

During our research, a few years ago, one of the things we found was that almost all MQTT implementations have the same behavior when it comes to duplicate Client IDs. The scenario is when a client connects to the server, if the client ID, presented by the new client that sent this CONNECT request, is already in use by another client that is currently connected to the server, the server goes ahead and disconnects the currently connected and lets the new client connect. There might be valid reasons for this behavior; hence it is not treated as a bug. However, this issue has the potential of causing DoS within the MQTT network. The information that is required is the logic for generating client IDs, and it becomes easy if they use a pattern. All you need in this case is a script that generates the client IDs and connects to the broker to effectively kick-out most if not all currently connected clients. In other words, cause an MQTT DDoS.

4.4.2 Client ID used for Authentication/Access Control

Sometimes (True story!) the client IDs might be used for authenticating a client or providing access to sensitive topics. In this case, using the same techniques as above, you can get access to the MQTT network and receive or manipulate sensitive data.

4.4.3 Cloning the Client

Again, depending on whether authentication is implemented or not and your access to credentials/keys of a legitimate thing, you can spoof the legitimate client and manipulate the telemetry data as if it was coming from the actual client or receive sensitive information destined for the legitimate client.

4.4.4 Attacking the Application via Malicious Input

Note: We have found and already disclosed a few vulnerabilities to known MQTT implementations using this technique. We are currently waiting for the disclosure timeline to finish so we can write a detailed chapter on it.

The IoT protocols were created for constrained environments. They are relatively new and the portability and data exchange with conventional protocols and applications is handled by the applications unless the protocol standard specifies the exchange mechanism. This is not a complex problem at all, but this opens a new attack surface. The Application developers typically trust the telemetry data coming from the device as the data is minimal in nature and may forget to filter it for known attacks. This means that depending on how the data will be processed on the application side, if the telemetry data contains a maliciously crafted payload, it can exploit the application. Let's take a very simple example of an ICS/IIoT Infrastructure where the temperature of a certain room is regularly sent to the application and the application displays it on the web UI. Assuming, the attacker has access to the MQTT network and can publish to the same topic, now publishes an XSS payload, this is bound to exploit the browser if the temperature data is not filtered by the application.

Disclaimer: This technique is basically applicable to any IoT protocol where telemetry data is transferred from the device to the cloud using the constrained IoT protocol (and maybe converting it to a conventional protocol like HTTP) and then fed to standard applications for storage, processing, or display without first filtering it.

4.4.5 Accessing and Manipulating Devices via messages

Using various techniques mentioned above, if you can receive sensitive messages (by subscribing to interesting topics) and/or are able to publish messages of your choice to sensitive topics, it is game over for the MQTT network. An example of publishing a message on a topic could be controlling the device by sending it specific commands on the topic that it expects to receivecommands from the Gateway/Cloud/User. That is scary!

4.5 Mitigations

Everything is not so bad with MQTT else, nobody would be using it in the first place. The vulnerabilities and issues in MQTT, as with any other protocol, stem from insecure configuration and use of the implementations. Some of the things that you should look at as a base-line for securing your MQTT implementation are:

1. Use TLS
2. Don't use Client IDs for authentication or topic access control.
3. 3. Use randomly generated Client IDs
4. 4. Do not expose `$SYS/` topics to clients. If required, use authentication based access control.
5. Treat Application Messages coming from the devices as tainted and filter them for known attacks based on the usage of that data.
6. Don't Hardcode credentials in the device firmware.
7. Devices (Clients) should have visibility or access to only the required topics.
8. Log all activity.
9. Prefer to log messages on `$SYS/` topics instead of publishing it on those topics and making it available for subscription, if it is not required. For e.g. Mosquitto server has a configuration option to log locally instead of publishing messages on any `$SYS/` topic.
10. Enable and use Cloud provider (AWS, Azure, etc.) security notifications for misbehavior or potential attacks in the MQTT infrastructure, for e.g. multiple failed Authentication attempts.

5. Conclusion

We hope this chapter gave you a good high-level overview of the MQTT protocol, how it works, and how it is used and how would you go about performing security assessments of MQTT based IoT/IIoT eco-systems. The tools, attacks, and techniques mentioned will improve assessment efficiency and give you a direction for a deeper inspection of security research on any MQTT implementation.

Chapter 5: MQTT Broker Security - 101

In this chapter, we are going to look at Client Authentication and other security methods used by MQTT broker for secure connections with MQTT clients

1. Introduction

MQTT protocol is becoming popular in many industries, it also fascinates IoT beginners, enthusiast IoT developers, and security researchers who want to explore this technology for new opportunities.

There are many open-source and commercial implementations for MQTT Broker and client available to choose from. But before using any MQTT implementation of MQTT broker, one must understand the security capabilities of MQTT broker and clients.

This chapter articulates the information required for beginners to start working on security of MQTT protocol and the security capabilities required for MQTT Broker and client.

We are using Eclipse Mosquitto Broker to understand broker configuration and how to configure the Mosquitto broker for desired security mode. We will also look into AWS IoT Core for similar features and configurations.

2. Data and Connection Security

2.1 Secure Connection

As per MQTT specs, TCP ports 8883 and 1883 are reserved for MQTT TLS and non-TLS communication, respectively. In non-TLS communication, i.e., on port 1883 , MQTT messages are transferred in plain-text format between MQTT broker and client and should be considered as an insecure connection. Whereas in TLS communication or secure connection i.e on TCP port 8883 , MQTT messages are encrypted by TLS security layer before transmission. It is always a good idea to use secure communication between MQTT broker and client.

2.1.1 Mosquitto Broker

Mosquitto Broker provides an option in `mosquitto.conf` file to select listener port for MQTT connection, default value is `1883` . It should be changed to `8883` for secure (TLS) communication.

```
# Port to use for the default listener.
port 8883
```

This is just the first step and you still need to configure broker for using certificates to make it work.

2.1.2 AWS IoT Core

AWS IoT core by default uses TLS connection for MQTT on port `8883`

2.2 Application data or MQTT Payload security

MQTT protocol is messaging protocol and does not provide any encryption for the application payload it is carrying. It is the responsibility of the application to encrypt / decrypt application data (payload) before sending to or after receiving from the MQTT layer to achieve end-to-end data encryption. Please note that application-level encryption is independent of TLS packet encryption.

3. Client Authentication

For any MQTT broker, there are three types of client authentication methods available to verify the identity of MQTT client:

1. Client Identifier (Client Id)
2. Username and Password
3. Client Certificates

3.1 Client Identifier (Client Id)

All MQTT clients must have a unique `Client Id` as per the MQTT Spec, and the client must send this `Client Id` with the `CONNECT` packet. The broker uses this `Client Id` to create and maintain the session state.

The `Client Id` is not part of MQTT authentication and is only used to uniquely identify the MQTT connection. However, some broker implementations provide some way to establish client identity using `Client Id` or something derived from `Client Id` . However, using `client ID` alone for authentication or verification is a bad idea because it can be manipulated/spoofed at the clientside.

We will see how Mosquitto Broker and AWS IoT Core use client id for basic security in the following examples

3.1.1 Mosquitto Broker

Mosquitto Broker provides an option called `clientid_prefixes` in `mosquitto.conf` file to configure Client ID prefixes, which allow clients with specified prefixes in their Client ID to connect to the Mosquitto broker.

```
# ======================================================================
# Security
# ======================================================================
# If set, only clients that have a matching prefix on their
# clientid will be allowed to connect to the broker. By default,
# all clients may connect.
# For example, setting "secure-" here would mean a client "secure-
# client" could connect but another with clientid "mqtt" couldn't.
clientid_prefixes C_ID
```

Here in the above example, clientid_prefixes is set to **C_ID**- and a client with `client id` let's say **C_ID_Client01** will be allowed to connect, but a client with the Client ID **Client01** will not be allowed to connect.

3.1.2 AWS IoT Core

AWS IoT provides the option to set `thing id` while creating `thing` in IoT core, here `thing` id is not the same as `Client ID` . However, AWS IoT allows connection for registered `thing` when the connection policy is configured for the corresponding `thing id` .

> Note:
> If the client with `client id` as `client_1` is connected and the second client with the same `client id` i.e. `client_1` is trying to connect the broker, then the first client is disconnected by the broker. This is the bare minimum security provided for any client session by MQTT protocol and keeps a client with intermittent connectivity from spawning multiple MQTT sessions. However, as discussed in our previous chapter on MQTT, it also opens the door for Denial of Service attacks on the MQTT network.

3.2 Username and Password

MQTT spec defines the requirement of a `username` and `password` for MQTT client authentication. A client sends the `username` and `password` with the CONNECT packet to the MQTT broker and the broker validates the `username` and `password` before accepting the MQTT session.

The `username` and password is sent in the `CONNECT` packet to the broker in cleat text format unless encrypted at the transport layer i.e. using port `8883` for connection.

> Note:
> According to the MQTT spec it is not mandatory to use authentication.

3.2.1 Mosquitto Broker

Mosquitto Broker provides two parameters in `mosquitto.conf` file to enable client authentication by client - `username` and `password` .

3.2.1.1 Allow Anonymous

(Please provide a sentence explaining what is allow anonymous)
To enable client authentication using credentials, you also need to set `allow_anonymous` parameter to `false` in `mosquitto`.conf file.

```
# Defaults to true if no other security options are set. If `password_file` or
# `psk_file` is set, or if an authentication plugin is loaded which implements
# username/password or TLS-PSK checks, then `allow_anonymous` defaults to
# false.
#
allow_anonymous false
```

3.2.1.1 Allow Anonymous

To create a password file, the mosquitto broker comes with mosquitto_passwd utility which creates a password file.

> Example:
> a. Create/append password file and add user `john` with hashed password

```
$ sudo mosquitto_passwd -c /etc/mosquitto/pwfile john <somepassword>
```

> b. Delete a user from a password file

```
$ sudo mosquitto_passwd -d /etc/mosquitto/pwfile john <somepassword>
```

Once password file has been created, set the current location of password file using `password_file` parameter in `mosquitto`.conf file.

```
# See the TLS client require_certificate and use_identity_as_username options
# for alternative authentication options. If an auth_plugin is used as well as
# password_file, the auth_plugin check will be made first.
password_file /etc/mosquitto/pwfile
```

> Note:
> The default location of the mosquitto password file for Linux is `/etc/mosquitto/pwfile` and for windows is the mosquitto installation folder `~\mosquitto\passwords.txt`

3.2.2 AWS IoT Core

AWS IoT core does not support username and password authentication, but it provides an option for custom authentication.

> Note:
> Another use of `username` is to allow or deny `topic` access to clients based on their `username`, we will see it in following section 4.1.3.2

3.3. Client SSL Certificate

Considered as the most secure method of client authentication, in certificate-based authentication client sends an SSL certificate, which is signed by a trusted root CA to the server to authenticate the client.

3.3.1 Mosquitto Broker

Mosquitto broker provides the below options in `mosquitto.conf` file to enable certificate-based client authentication.

3.3.1.1 Client Certificate require

When `require_certificate` is set to `true`, MQTT clients that provide valid certificate during connection handshake are allow to connect.

```
# By default, an TLS enabled listener will operate in a similar fashion to a
# https enabled web server, in that the server has a certificate signed by a CA
# and the client will verify that it is a trusted certificate. The overall aim
# is encryption of the network traffic. By setting require_certificate to true,
# the client must provide a valid certificate in order for the network
# connection to proceed. This allows access to the broker to be controlled
# outside of the mechanisms provided by MQTT.
require_certificate true
```

3.3.1.2 Root CA file Path

The `cafile` or `capath` must be set to a trusted CA certificate that has signed your server certificate, to enable certificate-based client authentication.

```
# At least one of cafile or capath must be defined to enable certificate based
# TLS encryption. They both define methods of accessing the PEM encoded
# Certificate Authority certificates that have signed your server certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificates.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must run
# "openssl rehash <path to capath>" each time you add/remove a certificate.
cafile /etc/mosquitto/certs/ca.crt
#capath
```

3.3.1.3 Server Certificate and keyfile Path

The `certfile` is requested by MQTT client for server authentication during TLS handshake.

```
# Path to the PEM encoded server certificate.
certfile /etc/mosquitto/certs/server.crt
# Path to the PEM encoded keyfile.
keyfile /etc/mosquitto/certs/server.key
```

3.3.2 AWS IoT Core

AWS IoT core uses x.509 certificate-based authentication as default client authentication method. Read more about AWS IoT client authentication here.

4. Restrict access to server resources

Every MQTT broker must have some access control mechanism to restrict MQTT clients from accessing server resources based on information provided by the client such as User Name, Client Identifier, the hostname/IP address of the client, or the outcome of authentication mechanisms.

4.1 Mosquitto Broker

Mosquitto broker has a mechanism called an `access control list` or `ACL` for MQTT topics, ACL is used to control the client access to subscribe and publish to topics on the broker. User name or Client ID is used to restrict access to broker topics.

The `acl_file` parameter in `mosquitto.conf` is a path to a file that contains an access control list. Valid file path enables `ACL` for topic restriction.

```
# If an auth_plugin is used as well as acl_file, the auth_plugin check will be
# made first.
acl_file /etc/mosquitto/aclfile
```

> Note:
> Location of default `ACL` file i.e `aclfile.example` for linux is `/usr/share/doc/mosquitto/examples/aclfile.example` and for windows it is the mosquito installation folder `~\mosquitto\aclfile.example.`

Default content of `aclfile.example` file

```
# This affects access control for clients with no username.
topic read $SYS/#

# This only affects clients with username "roger".
user roger
topic foo/bar

# This affects all clients.
pattern write $SYS/broker/connection/%c/state
```

There are three types of rules by which `ACL` is established and saved in `acl_file`

4.1.1 topic name

This rule uses the `topic` keyword and is applicable to all anonymous clients. If a client uses credentials for the connection, then this rule is ignored for that client.

Below is the format used to define this rule -

```
topic [read|write|readwrite] <topic>
```

> Example:
> Below line gives read (subscribe) access to any anonymous client and access to any other topic would be denied.

```
topic read $SYS/#
```

4.1.2 user name

This rule uses the `user` keyword and is for the clients that use user credentials for the connection.
Below is the format used to define this rule -

```
user <username>
topic [read|write|readwrite] <topic>
```

Here the `username` is the same as in the password file

> Example:
> Below lines gives read (subscribe) and write (publish) access to topic `foo/bar` to a client with user name `roger` and ignored for other clients.

```
user roger
topic foo/bar
```

4.1.3 pattern substitution with the topic

This rule uses the `pattern` keyword. Using pattern substitution with the topic, a broker allows access to a client using its `client id` or `username` .
Below is the format used to define this rule -

```
pattern [read|write|readwrite] <topic>
```

> Note: As per Eclipse Mosquitto documentation following patterns are available for substitution
> 1. %c to match the client id of the client
> 2. %u to match the username of the client

The following line is the default entry in the ACL file, and grants write (publish) access to all clients.

```
pattern write $SYS/broker/connection/%c/state
```

4.1.3.1 Client ID substitution

Following example will illustrate how to use `client id` substation for `topic` access

> Example:
> Below line allow read access to topic `pattern write sensor/<clientid>/data` to all clients

```
pattern write sensor/%c/data
```

> Below line will allow write access to client with client id c1 to access this topic and denies access to all other clients

```
pattern write sensor/c1/data
```

4.1.3.2 User name substitution

Following example will illustrate how to use `username` substation for `topic` access

> Example:
> Below line allow read access to topic `user/<username>/logintime` to all client

```
pattern read user/%u/logintime
```

> Below line will allow read and write access to the topic `user/john/lastlogin` to user `john`

```
pattern read | write user/john/lastlogin
```

4.2 AWS IoT Core

In AWS IoT, permissions to access the broker's resources are controlled by `AWS IoT Core policies`. Read more on AWS IoT Core policies here.

5. Certificate Revocation List

Though certificate-based authentication is considered as most secured, certificate once issued, it cannot be modified, and controlling one's access when it is not expired is a challenge. And then, there are scenarios where certificates form cloned or decommissioned devices, are used to get system access. To avoid such scenarios, Certificate Revocation List (CRL) is issued by certificate authority CA. It contains information on the revoked certificates from decommissioned or compromised devices that no longer be trusted and prevent from any future use.

5.1 Mosquitto Broker

The `CRL` file is a must when certificate-based client authentication is used i.e. `require_certificate` set to `true`. It contains information of the revoked certificates from decommissioned or compromised client devices to prevent any future use.

You can use the `crlfile` parameter from `mosquitto.conf` to point to the PEM encoded revocation file.

```
# If you have require_certificate set to true, you can create a certificate
# revocation list file to revoke access to particular client certificates. If
# you have done this, use crlfile to point to the PEM encoded revocation file.
crlfile /etc/pki/mosquitto/content/crl/mosquitto_crl.pem
```

5.2 AWS IoT Core

Using AWS IoT console, the user can deactivate or revoke the client certificates. Read more on client certificate activation or deactivation here.

6. Conclusion

There are serval security mechanisms available out there to secure MQTT communication, and it varies from implementation to implementation. These security mechanisms implemented at the MQTT broker side, correct configuration of MQTT broker and MQTT client reduce the attack surface in an IoT ecosystem and give a better edge over known attacks. We hope this chapter gave you a good insight into the security mechanisms used to secure an MQTT connection between a broker and a client.

COAP

Chapter 6: Introduction To CoAP Protocol And Security

In this chapter, we are going to look at CoAP, a simple IoT protocol used for constrained environments, and security issues and attacks on CoAP protocol and its implementations. The power of CoAP is its simplicity and portability with HTTP protocol, which will be evident when we discuss the protocol internals.

1. Introduction

MQTT protocol is becoming popular in many industries, it also fascinates IoT beginners, enthusiast IoT developers, and security researchers who want to explore this technology for new opportunities.

UDP (or TCP) Port	Communication Type
5683	Plain text
5684	DTLS (or TLS over TCP)

1.1 CoAP Features

1. It provides a simple discovery mechanism
2. Integration with Web is easy
3. It provides asynchronous message exchange
4. Uses URIs to define resources/services
5. Uses REST-like request/response model

1.2 HTTP Lineage

CoAP was designed as a generic web protocol for constrained environments. It utilizes parts of HTTP, more specifically the REST architecture, in a compressed binary format. It can be easily integrated with the Web owing to simple translation/proxying between CoAP and HTTP. Think of it as a Poor man's HTTP. If you look at the HTTP protocol, it is quite extensive and Chatty owing to its text-based semantics. When working with constrained devices, chatty protocols are not a good choice for long battery life and processing power. You will get a better idea when we discuss CoAP architecture and communication.

1.3 Use and Popularity

From our experience with IoT devices, CoAP functionality is provided on the RF side in various RF protocols. It is not quite as popular as MQTT for Ethernet/WiFi-based devices. However, you may still end up finding products that use CoAP for communication over WiFi/Ethernet.

2. CoAP Communication

2.1 Protocol Overview

CoAP uses the Client-Server communication model where nodes send requests and receive responses from other nodes. It uses a two-layer approach for communication where one layer deals with the UDP protocol and the other deals with the application data:

1. CoAP Messages - These define the type of CoAP packets and deal with UDP.
2. Requests/Response (Similar to HTTP) - These encapsulate the actual application payload/data using Request Methods and Response Codes.

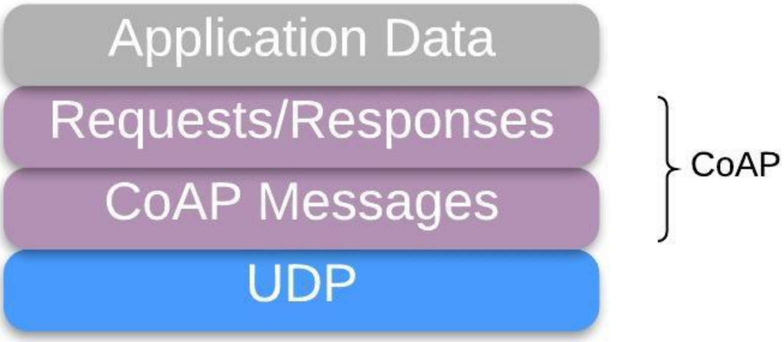


Fig 1. CoAP Protocol Layers

Image Source : https://payatu.com/blog/aseem/iot-security---part-11-introduction-to-coap-protocol-and-security

As we can see, CoAP uses UDP for message transfer, and encapsulates the request/response, application data in the messages. Since it uses UDP, it has to manage the reliability part. It does that using ACK messages.

1.1 CoAP Messages

The CoAP standard defines 4 different types of Messages:

1. Confirmable Message (CON) - This type of message requires that the receiver send an Acknowledgment message back to the sender for confirmation of the receipt of this message
2. Non-Confirmable Message (NON) - Does not require any ACK
3. Acknowledgment Message (ACK) - This message is sent, by a receiver, as an Acknowledgment for a Confirmable message that is received from the sender

4. Reset Message (RST) - This is message is typically sent when the receiver is not able to process a Confirmable or Non-Confirmable message due to some error.

Each message contains a 16-bit message ID to uniquely identify a message and map its corresponding ACK if any. This is typically used for identification and deduplication of messages as with other protocols as well. Below are some simple sequence diagrams that explain basic message exchange:

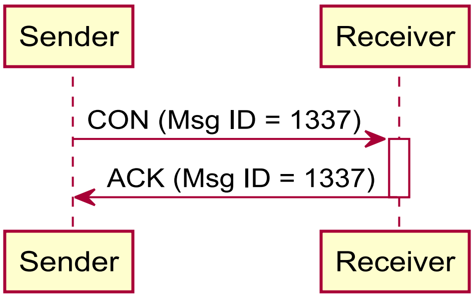


Fig 2. Confirmable Message Communication

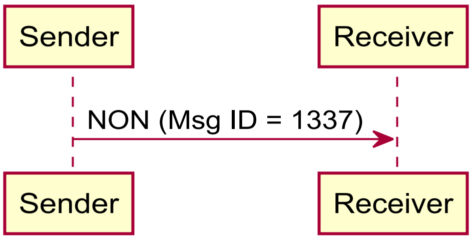


Fig 3. Non-Confirmable Message Communication

2.1.2 CoAP Request/Response

The Messages can contain a request, response, or maybe empty. There is no relation between the request/response and the message type other than the fact that some combinations are not possible or define some specific purpose. Let's first look at the Request and Response format and then delve deeper into the packet format.

1. Requests use Method Codes and Responses use response codes similar to HTTP but defined in binary format.
2. CoAP standard defines four Request methods - GET, PUT, POST, and DELETE.
3. CoAP standard defines Response code similar to HTTP - 2.xx for success, 4.xx for client error, and 5.xx for server error.
4. Unique 0-8 byte Tokens are used for identifying, mapping request and response much the same way Message-IDs are used to identify messages and map ACKs.

Let’s look at some examples of request/response communication to get a better idea about how applications exchange data. The first figure shows a request in a Confirmable message with the response piggybacked in the Acknowledgment Message.

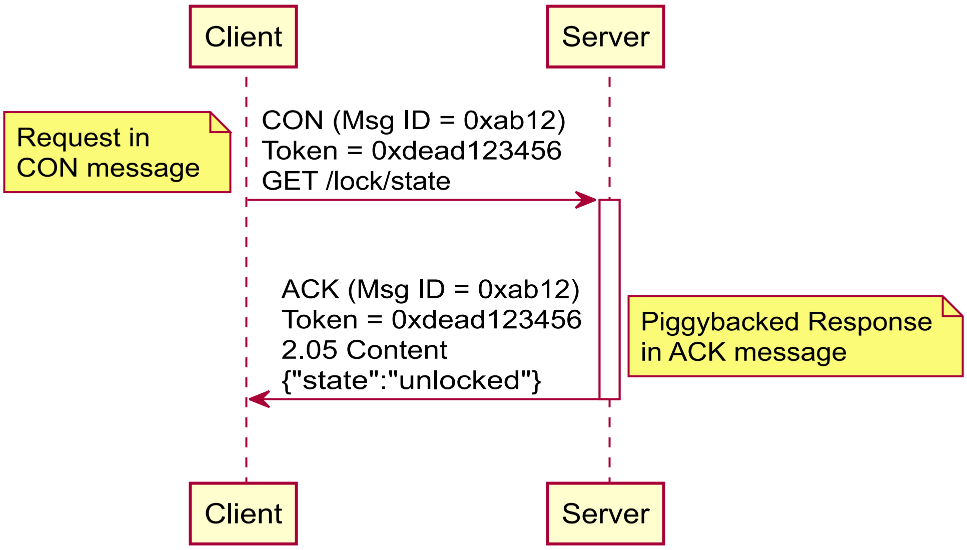

Client
Server
Request in CON message
CON (Msg ID = 0xab12)
Token = 0xdead123456
GET /lock/state
ACK (Msg ID = 0xab12)
Token = 0xdead123456
2.05 Content
{"state":"unlocked"}
Piggybacked Response in ACK message
Client
Server

Fig 4. CON Message Request with Piggybacked Response

In the below figure, you can see the lifetime and mapping for Message-IDs and Tokens when request and response are sent in separate Confirmable messages and the Acknowledgment Messages are empty.

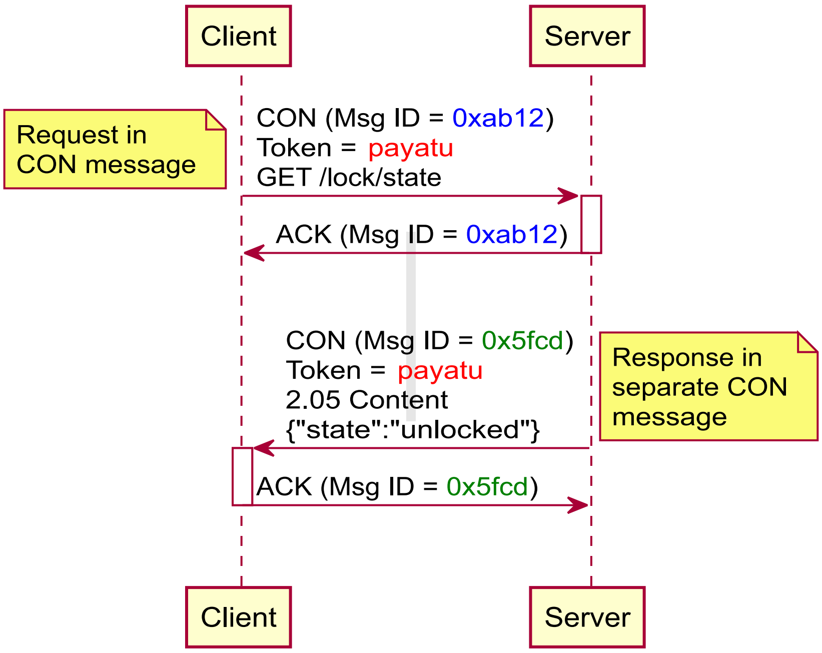

Client
Server
Request in CON message
CON (Msg ID = 0xab12)
Token = payatu
GET /lock/state
ACK (Msg ID = 0xab12)
CON (Msg ID = 0x5fcd)
Token = payatu
2.05 Content
{"state":"unlocked"}
Response in separate CON message
ACK (Msg ID = 0x5fcd)
Client
Server

Fig 5. Separate CON Messages for Request and Response

The below figure shows an example when requests and responses are sent in separate Non-Confirmable messages.

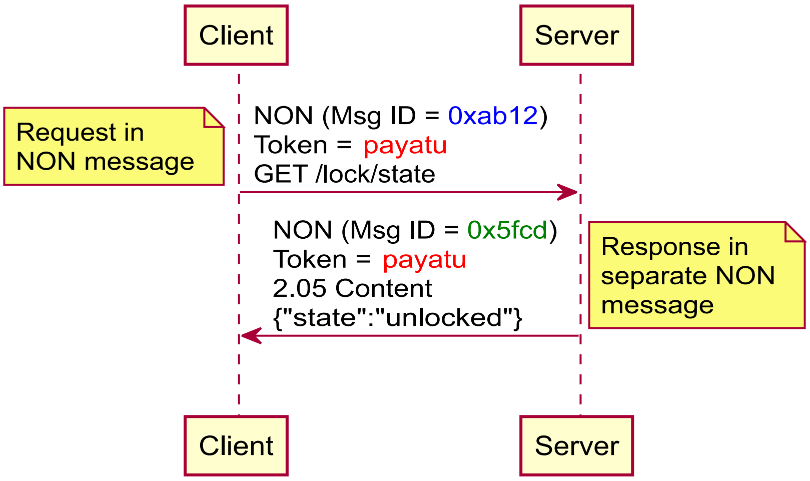

Client
Server
Request in NON message
NON (Msg ID = 0xab12)
Token = payatu
GET /lock/state
NON (Msg ID = 0x5fcd)
Token = payatu
2.05 Content
{"state":"unlocked"}
Response in separate NON message
Client
Server

Fig 6. NON Message Request and Response

Of course, there are combinations of Message and Request/Response which are not allowed by the standard or have a specific meaning. The below table shows what is possible and what is not.

	CON	NON	ACK	RST
Request	Yes	Yes	No	No
Response	Yes	Yes	Yes	No
Empty	CoAP Ping	No	Yes	Yes

CoAP ping, as suggested in the protocol, can be used for heartbeat/health check of the server. A CoAP ping communication works as below:

1. The sender sends an empty Confirmable message.
2. The receiver responds with an empty Reset message.

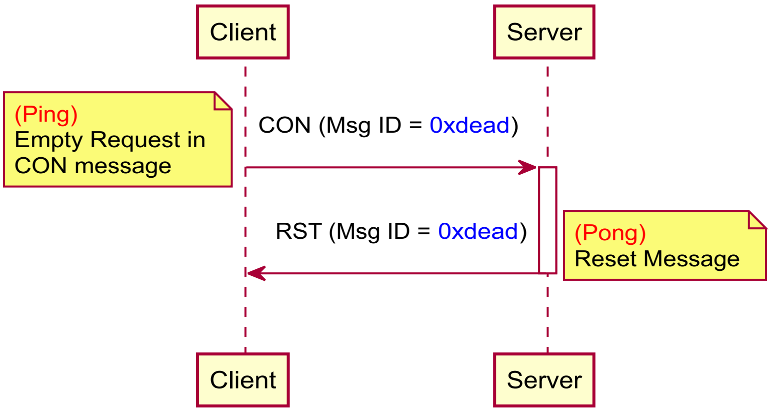

Client
Server
(Ping)
Empty Request in CON message
CON (Msg ID = 0xdead)
RST (Msg ID = 0xdead)
(Pong)
Reset Message
Client
Server

Fig 7. CoAP Ping Communication

2.1.3 CoAP Request and Response Codes

The requests and responses are identified by codes. The response codes are derived from HTTP. The code field is divided into two parts:

1. 3-bit class
2. 5-bit detail

The below tables describe the codes, denoted as c.dd format where c is the class and dd is the detail:

Request Codes

Code	Description
0.00	Empty message (Special Case)
0.01	GET
0.02	POST
0.03	PUT
0.04	DELETE

Response Codes

Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method Not Allowed
4.06	Not Acceptable
4.12	Precondition Failed
4.13	Request Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported

2.1.4 CoAP Options

You might be thinking that we have discussed a lot about the Request/Response model and how CoAP is similar to HTTP. But there is still not much resemblance with HTTP other than the request and response line, right? This section talks about the Options field of the CoAP protocol, which is somewhat similar to HTTP headers, albeit in binary format. Options define metadata that is sent to the other endpoint, much like the data that is sent using headers such as host, content type, etc. Each Option field defines the option number as defined by the standard, its length, and its value. Let's look at some of the options defined in the standard, which might sound familiar:

No.	Name	Format	Length(Bytes)	Description
3	Uri-Host	String	1-255	The hostname/IP address of the CoAP Server on which the resource is being requested
7	Uri-Port	uint	0-2	The UDP port no. of the CoAP server
11	Uri-Path	string	0-255	Each Uri-Path specifies a single segment of the absolute resource path of the URI
12	Content-Format	uint	0-2	ID of the content format of Application data (IDs are defined by the standard)
15	URI - Query	string	0-255	Each Uri-Query specifies single parameterized argument in the query part (query string) of the URI
17	Accept	uint	0-2	Indicates which Content-Format is acceptable to the client.

Following are the Content-Format types and their respective IDs defined by the standard:

Type	ID
text/plain;charset=utf-8	0
application/link-format	40
application/xml	41
application/octet-stream	42
application/exi	47
application/json	50

2.2 Discovery Mechanism

As with many other IoT protocols, CoAP also has a discovery mechanism. The discovery mechanism a very important aspect of m2m communication that aids automation without the need for human interaction. CoAP discovery is geared towards identifying the resources available on a CoAP server. It is supported using the **Co**nstrained **RE**STful Environments (CoRE) Link Format protocol RFC 6690. Cutting the long story short a CoAP Link Format of resources is supported on a CoAP server and queried by a client to identify the resources available on the server. This is performed by sending a GET request on `/.well-known/core` resource path as specified in the standard. When a server receives the request, it responds with a list of all the resource URIs available on the server.

Given below is an example taken from the standard itself. The response contains the commaseparated

list of resource URI entries available on the server. Each entry has:

1. The resource URI in angular brackets ex. `</sensors>`
2. URI attributes separated by semicolons `;` ex. `title` , `rt` , `if` .

Request

`GET coap://server:port/.well-known/core`

Response

```
2.05 Content
</sensors>;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"
```

3. CoAP Security

Now that we understand how CoAP works, let's focus on the security aspect of CoAP both from an offensive as well as defensive perspective. All the information below is based on our experience with CoAP security research and IoT product and infrastructure penetration testing projects. Below we will try to answer some of the questions that generally come up when one encounters a new protocol in their IoT penetration tests.

3.1 Protocol Security/Authentication

We have discussed the protocol at length without a single mention of authentication. Surprised? The CoAP standard does not specify any authentication mechanism. It relies on DTLS to provide protocol-level security. The standard defines four security modes for devices post provisioning:

1. `NoSec` - This means there is no protocol-level security i.e. DTLS is disabled.
2. `PreSharedKey` - In this mode, DTLS is enabled and the device has a list of pre-shared keys with each key, including a list of which nodes it can be used to communicate with.
3. `RawPublicKey` - In this mode, DTLS is enabled and the device has an asymmetric key pair without any certificate. The key is validated using an OOB (Out-of-bound) mechanism
4. `Certificate` - In this mode, DTLS is enabled and the device has an asymmetric key pair along with an X.509 certificate that is signed by a common and trusted Root CA.

The protocol standard clearly mentions in section 9 (page 69) - "CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload)" There is another standard RFC 8613 that defines Object Security though.

3.2 Tools

1. Open-source libraries, clients, server implementations. The only issue here is that they are rigid and have no guidance from the security perspective and you may end up editing or writing custom scripts to do the job. The list of all libraries and packages can be found here.
2. EXPLIoT Framework We are planning to release basic plugins for CoAP assessment in the next couple of months.

3.3 Recon

Ok, so we have the tools, now what? Where do we start? The first thing is to perform recon on the CoAP Server to get as much information about the server as possible.

1. Is there any Authentication mechanism implemented?
2. Get a list of available resources on the server by sending a GET request to `/.wellknown/core` URI path.
3. What do the URIs signify? Do their names indicate any action related to the physical environment (sensor/actuator)? Is there any URI for configuration/setting update?
4. Are you able to read (GET) any sensitive information?
5. Are you able to manipulate (PUT/POST/DELETE) any sensitive information?
6. What URIs are important and used in the IoT Eco-system? This can also be found by reverseengineering the device firmware or Mobile App (if it communicates with the sensor).
7. Is DTLS used?
8. How is the telemetry data, sent from the device, utilized on the cloud?

3.4 Analysis

Once we gather the basic information about the CoAP implementation, we can utilize it to launch different attacks on the devices or the eco-system. By eco-system, I mean all the components of the IoT product infrastructure including the devices, gateways, Cloud and apps, mobile apps, etc. We will list down some of the ideas to get your grey cells working:

1. Brute-forcing/Fuzzing all URIs with different Request methods to identify which URIs respond to which methods. However, there is a risk of corrupting the data on the device using this method. Also, a bit more information about the query parameters and application data format would help in directed fuzzing to get meaningful results
2. If there is custom authentication implemented, you will need to reverse engineer it to identify the mechanism.

3. Reversing the firmware or the mobile app may give you sensitive information such as
    1. Resource URIs
    2. Parameters
    3. Methods
    4. Application Data Semantics.
    5. What data is exchanged and expected on specific URIs and how it affects the behavior of the components.

3.5 Attacks

3.5.1 Path Traversal (Old wine in new bottles)

uring our research, a couple of years ago, one of the things we found was that few CoAP implementations are not (or incorrectly) parsing or ignoring double dots ".." in the URI. The issue here is the same as with Web, for example, if the URI is bound to a file, there are chances of breaking out of CoAP root and accessing or manipulating other files. There might be other issues in file-less URIs that researchers may find in the future, who knows. Below are direct quotes from the Standard about the dots:

1. Section 5.10.1 page 54 : "The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options)."
2. Section 5.10.7 page 57 - "Each Location-Path Option specifies one segment of the absolute path to the resource, and each Location-Query Option specifies one argument parameterizing the resource. The Location-Path and Location-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Location-Path Option MUST NOT be "." or ".."."

There are a few important things to note here:

1. 1. No percent-encoding is performed.
2. The request URI must be resolved before parsing it into options.
3. The path options must not contain "." or ".."
4. Any CoAP compliant library/implementation should adhere to what is mentioned in the standard.
5. It does not mention how to handle the case when the value of the Option in the received packet is either "." or "..".

Based on the above information, lets quickly look at two implementations where we checked for double dots:

1. Californium - Californium is a well-known java-based CoAP implementation. The core implementation of Californium does not parse/ignore double dots. That is left to the users to implement. In one of their demo (sample) apps called cf-simplefile-

server, they have implemented a method `checkFileLocation()` to detect this attack. It means that the core protocol implementation of californium does not provide any security against this. We sent an email to one of the core developers of californium about this on 26th Feb 2019. As usual, there was no response. We did not pursue it further, if someone files a bug and manages to get a CVE, please do add our name in the credits :).

2. `libcoap` - is also a famous C implementation of the CoAP protocol. They do have a provision for filtering dots. They have implemented a function `dots()` in the code. However, if the path is percent-encoded the check is bypassed. The issue is that the function responsible to do this `coap_split_path_impl()` checks for the dots first and then passes it to another function `h()` where the path is percent-encoding handling as can be seen in the below code snippet from the file uri.c

```
if (!dots(p, q - p)) {
h(p, q - p, data);
}
```

An example would be: `GET coap://coap_server/foo/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd`
We sent the author an email on the same day as Californium i.e. 26 Feb 2019. In this case, the author replied, however, the response was that it will work only if the server implementation isbroken. Again, we did not pursue it further as the author did not agree that it is an issue on their side.

3.5.2 Cross Protocol Attacks

The CoAP protocol has tight coupling with HTTP in many fronts including request/response format, caching, proxying, etc. This also gives rise to opportunities for HTTP-to-CoAP translation/proxying and vice versa. Due to the similarity, some of the traditional attacks that affect HTTP today could very well apply to CoAP now or in the future. As we have already seen in the above case we discovered Path Traversal issues in CoAP implementations, however, they were originally discovered in HTTP. It's only a matter of time when researchers start connecting the dots and finding interesting attacks that target headers/options, URIs, request/response, etc. by way of either:

1. Finding it feasible to attack CoAP with traditional attacks
2. Finding ways to transfer an attack from one protocol to the other.
   Note: By Cross Protocol Attacks we mean attacking the protocol and not the high applicatio nwhich we discuss in the below scenario.

3.5.3 Attacking the Application via Malicious Input

In the CoAP application architecture, typically the server resides on the sensors and you can discover the resources/services available on the server and send the request. There are two places where we can attack the applications:

1. Server i.e. the application running on the sensor - This simply means sending it requests with malicious Application payload. You can do it blindly by fuzzing the payloads or a better approach is to reverse engineer the firmware and get a better understanding of what application data format is being used, how do the URI queries look like and then craft your malicious inputs.
2. Client i.e. the application running on the Cloud/Mobile - The idea behind this is quite similar to what we discussed in the previous chapter on MQTT. The premise of this attack is the fact that cloud/mobile application developers may not filter the input from a sensor for known application attacks and this can lead to exploiting those apps using traditional attacks like SQLi, XSS to name a few.

3.5.4 Accessing and Manipulating Devices Via Requests

Using various techniques mentioned above, if you can read sensitive data and/or can write data of your choice to sensitive resources by any of the request methods, you have complete control over the device. An example could be controlling the device by sending it specific commands that it expects to receive from the Gateway/User. Controlling or shutting down an IoT device performing a critical operation in itself is game over!

3.5.5 Cloning or MitM'ing the Server (Sensor)

Depending on whether the authentication/security is implemented or not and your access to the keys, Certificates, authentication credentials if any, you can have a rogue device pretend to be a Sensor and publish all the resources available on the legitimate sensor, you can send wrong or malicious data to the client.

3.5.6 Distributed Denial of Service using CoAP

The fact that CoAP runs over UDP makes it a perfect choice for attackers to cause DDoS on targetted nodes or machines within the network as well as on the Internet. In fact, utilizing CoAP devices available on the Internet for DDoS by malicious actors is already a known thing and attackers are doing it as we speak. The science behind this is pretty old and common - Attacker spoofs the IP of the victim and sends UDP datagram (CoAP request in this case) to the Server, which then responds to the victim IP. If you send mass requests (Spoofed IP) to all available servers on the network/Internet, they will in-turn respond to the victim and overload its kernel thereby causing Denial of Service on the victim.

3.6 Mitigations

The vulnerabilities and issues in CoAP are quite similar to other protocols, lack of authentication, access control. There are a few things that one should consider when trying to implement a secure CoAP service:

1. Use DTLS.
2. Prefer not to implement a custom authentication/encryption mechanism. History tells us that custom security implementations seldom work as desired :).
3. Prefer to utilize all 8-byte to generate random Tokens for Requests. It's good to utilize the max. length of the token to generate as much randomness as possible as smaller the size, easier it would be to brute-force/guess and spoof.
4. 4. Filter double and single dots (".." and ".") in the Uri-Path to prevent against path traversal attacks.
5. Secure the keying material/Certificates etc properly on the devices.
6. Filter input (Request Application payload) on the Server (Device) side.
7. Filter input (Response Application payload) on the cloud side. Even though the device is sending simple telemetry data, it is good to filter it for known/traditional attacks.
8. Implement proper access control and Auth mechanism for accessing resources that perform a critical operation or provide critical information.
9. Log all Activity.
10. Have a way to notify cloud/user about unauthenticated/malicious looking requests.
11. Don't hardcode credentials/sensitive information in the device firmware.

4. Conclusion

We hope this chapter gave you a good high-level overview of CoAP protocol, how it works, how it is used, and how you could go about performing security assessments of a CoAP based IoT/IIoT ecosystem. Some of the attacks mentioned in the chapter may have not been exploited in the wild or reported to the vendor as the protocol, and these techniques are fairly new. This is your chance to find as many vulnerabilities and new attack vectors as possible. The tools, attacks, and techniques mentioned above will improve assessment efficiency and give you a direction for deeper inspection and security research on any CoAP implementation.

BLE
Chapter 7: Bluetooth Low Energy – 101

Bluetooth has been a buzz-word as people wanted all their devices to be smart and which basically implies that you get to control things across the devices and not needing to carry wire around. Bluetooth has been in the market for more than a decade. If you're a millennial, you would have used those classic fancy Nokia phone which has Bluetooth in it. Bluetooth was invented by Ericsson and other vendors have started using Bluetooth. Soon after that, all the major vendors created a consortium called as Bluetooth Special Interest Group – SIG, which governs how the standard should be and the interoperability between different versions.

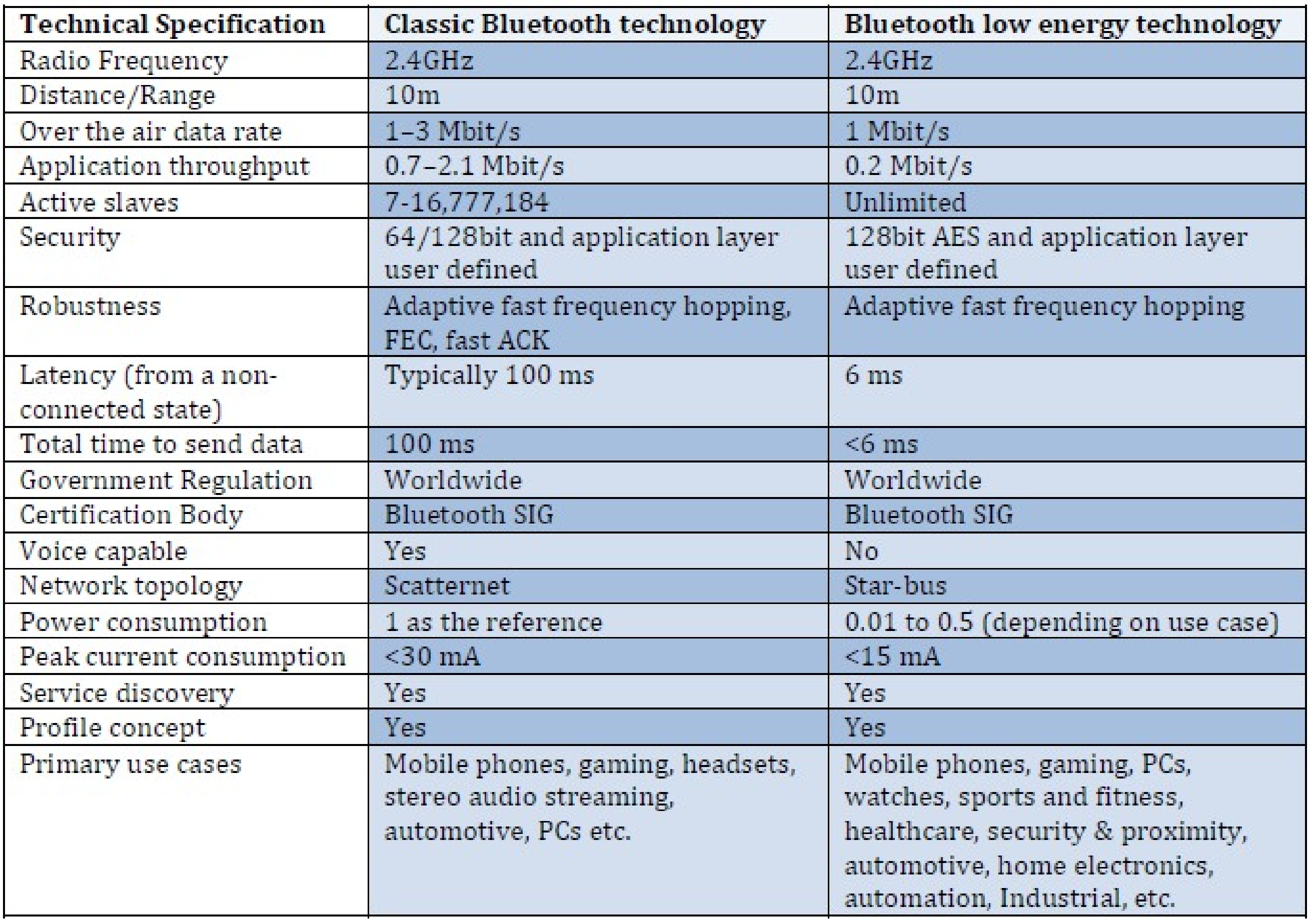
We are not going to talk about Bluetooth. Bluetooth by itself is a massive stack and their specification is around 2000+ pages. In this chapter, I will be covering only the Bluetooth Low Energy, more famously known as BLE.

With the advent of connecting all the things to the internet, there comes the problem of power and resource. As I mentioned early, Bluetooth is a huge stack. Implementing it in an end device like a fitness band would take more power and resource. So in the Bluetooth 4.0 standard, they introduced something called Low energy, which is specially targeted for IoT and smart devices that runs on memory and power-constrained devices.

Bluetooth®
SPECIAL INTEREST GROUP

Bluetooth SIG started selling the standard as Bluetooth Smart, which has two components. Bluetooth smart devices are end devices, that have only the Bluetooth Low Energy component and Bluetooth smart Ready are the device which is capable of doing both the Bluetooth LE and the EDR-Bluetooth classic component which could be your central device, i.e, mobile phone or laptop.

Now let's look into the technical details of the Bluetooth specification



Technical Specification	Classic Bluetooth technology	Bluetooth low energy technology
Radio Frequency	2.4GHz	2.4GHz
Distance/Range	10m	10m
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.2 Mbit/s
Active slaves	7-16,777,184	Unlimited
Security	64/128bit and application layer user defined	128bit AES and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive fast frequency hopping
Latency (from a non-connected state)	Typically 100 ms	6 ms
Total time to send data	100 ms	<6 ms
Government Regulation	Worldwide	Worldwide
Certification Body	Bluetooth SIG	Bluetooth SIG
Voice capable	Yes	No
Network topology	Scatternet	Star-bus
Power consumption	1 as the reference	0.01 to 0.5 (depending on use case)
Peak current consumption	<30 mA	<15 mA
Service discovery	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, gaming, headsets, stereo audio streaming, automotive, PCs etc.	Mobile phones, gaming, PCs, watches, sports and fitness, healthcare, security & proximity, automotive, home electronics, automation, Industrial, etc.

Source:
https://archive.eetindia.co.in/www.eetindia.co.in/STATIC/ARTICLE_IMAGES/201312/EEIOL_2013DE C13_RFD_NET_TA_01Tab1.gif

The table itself will give you a better insight into the specification, range and bandwidth has been reduced to withstand the low power and low resource.

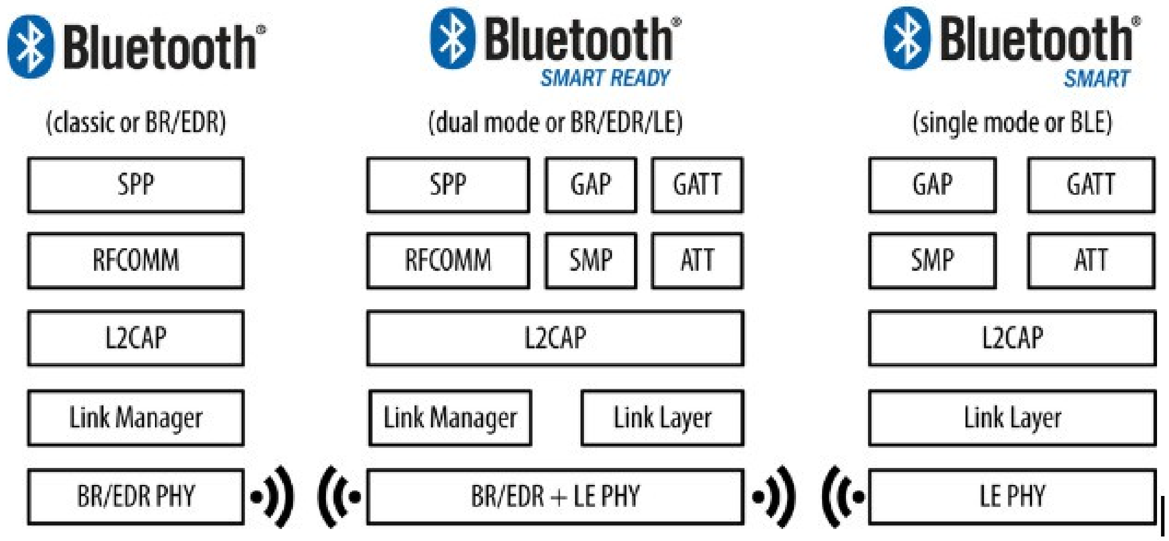
As I mentioned earlier, LE has two different types of devices.

Bluetooth Smart Ready – Which are the central device which is battery powered and high resource which is capable of running all the Bluetooth protocols. They are your laptops and a mobile phone.

Bluetooth Smart - They are your end devices like fitness tracker or baggage tracker or a smart dildo. They don't have to run an entire stack and they need to conserve power and resource. They run only the Bluetooth LE server. They are the peripheral device that the central device can connect to.

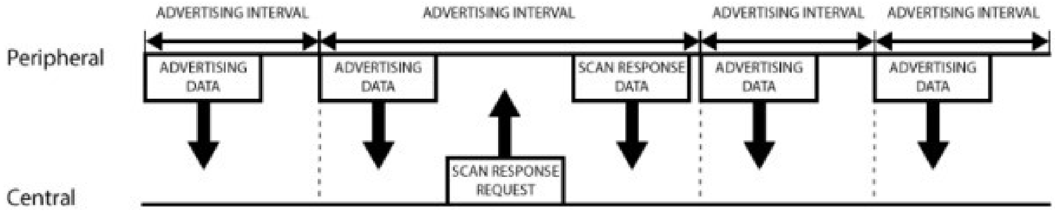
Bluetooth and LE stack details are out of the scope of this document.

But the two important components we will focus on are GAT and GAPP, which are responsible for the operation of the BLE service.


Bluetooth
(classic or BR/EDR)
SPP
RFCOMM
L2CAP
Link Manager
BR/EDR PHY
Bluetooth SMART READY
(dual mode or BR/EDR/LE)
SPP
GAP
GATT
RFCOMM
SMP
ATT
L2CAP
Link Manager
Link Layer
BR/EDR + LE PHY
Bluetooth SMART
(single mode or BLE)
GAP
GATT
SMP
ATT
L2CAP
Link Layer
LE PHY

1. Generic Access Profile (GAP)

GAP defines how your communication and connection to the central and peripheral should work.


ADVERTISING INTERVAL
ADVERTISING INTERVAL
ADVERTISING INTERVAL
ADVERTISING INTERVAL
Peripheral
ADVERTISING DATA
ADVERTISING DATA
SCAN RESPONSE DATA
ADVERTISING DATA
ADVERTISING DATA
SCAN RESPONSE REQUEST
Central

Source:https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

2. Generic Attribute (GATT)

GATT is like a server that manages how your data needs to be treated. Your Bluetooth LE devices work as a server-client principle. Here your end device/peripheral device acts as the server which runs the GATT server, and your central device acts as the client. So your end app or the tool connects to the GATT server and requests data from the device. Inside your GATT server, there are three components.

2.1 Profile

(http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gatt.html)
Which is defined by the Bluetooth SIG, it could be based on the type of the device, be it a blood pressure device or temperature sensor or any most commonly used device which has an advantage of interoperability.

2.2 Services

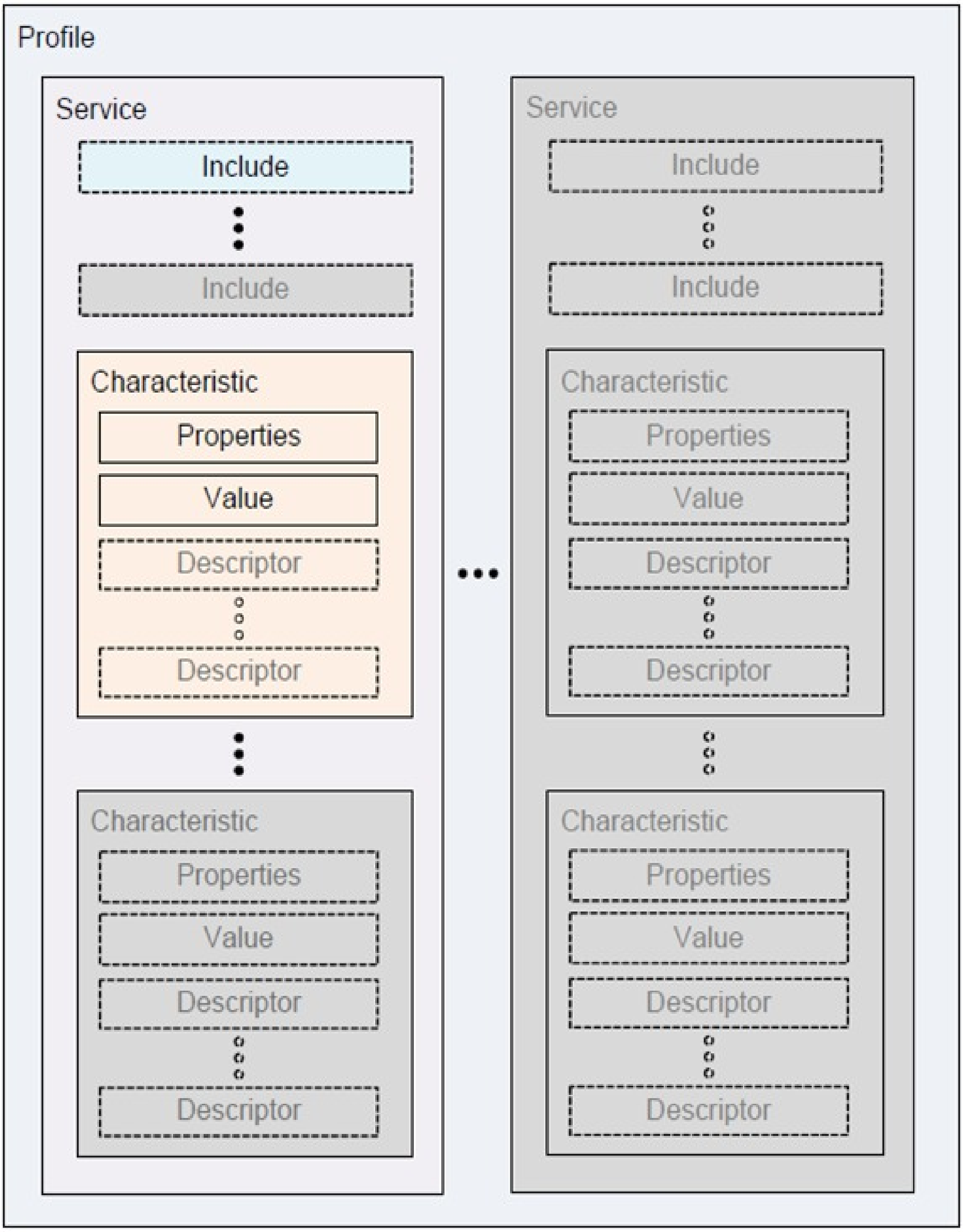
(https://www.bluetooth.com/specifications/gatt/services) –
Each device has multiple parameters inside it. Let's say a device could have a name, firmware version, OTA functionality, device operation. They are grouped into their specific datasets called as service.

2.3 Characteristics

It is found inside your service is where your data is placed. It could be a 16 bit Bluetooth SIG derived characteristic or a vendor-specific 128-bit characteristic.

Source: https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

In short, service is like a folder and characteristics are the files which holds the data.

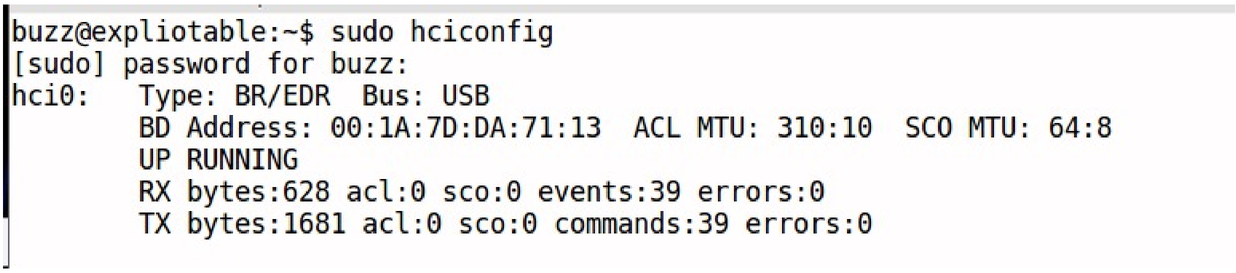

Profile
Service
Include
Include
Characteristic
Properties
Value
Descriptor
Descriptor
Characteristic
Properties
Value
Descriptor
Descriptor
Service
Include
Include
Characteristic
Properties
Value
Descriptor
Descriptor
Characteristic
Properties
Value
Descriptor
Descriptor

Now that we understood the basics of what Bluetooth LE is and how it functions. Let's go into some tools and methods on how to access the BLE devices.

If you are using windows, I would seriously suggest you use Ubuntu as it comes with all the necessary tools to access BLE devices and get those cheap Bluetooth 4.0 dongles from Amazon. (some laptops don't come with it.)

2.4 Connecting your Bluetooth dongle:

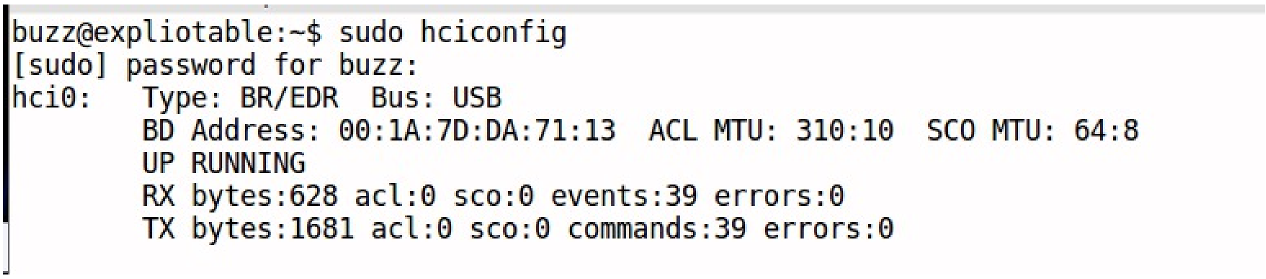
1. Connect the Bluetooth USB Dongle to the free USB port of your laptop. (No need to install any driver from your host machine).
2. Once Connected, open your terminal and type “**sudo hciconfig**“ You should be able to see this window which gives you the mac address(The USB dongle) and it should say UP and RUNNING.
3. If you encounter any issue restart the Bluetooth interface by **“sudo hciconfig hci0 reset”**– This will be handy a lot of time.



```
buzz@expliotable:~$ sudo hciconfig
[sudo] password for buzz:
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING
        RX bytes:628 acl:0 sco:0 events:39 errors:0
        TX bytes:1681 acl:0 sco:0 commands:39 errors:0
```

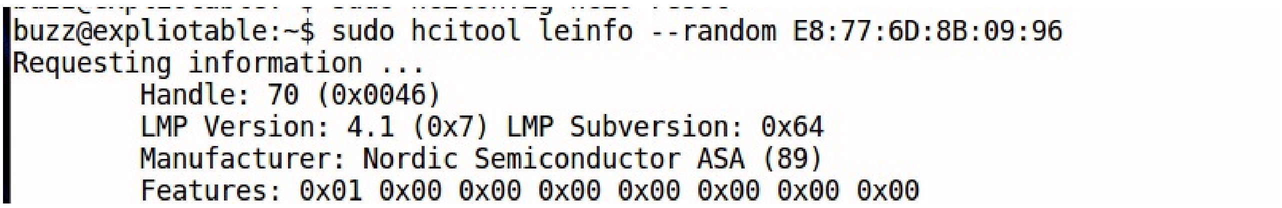
2.5 Scanning for Bluetooth devices

1. Once you have successfully connected your Bluetooth dongle to your machine
2. You can now scan for all the BLE devices around you using “**sudo hcitool lescan**”



```
buzz@expliotable:~$ sudo hciconfig
[sudo] password for buzz:
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING
        RX bytes:628 acl:0 sco:0 events:39 errors:0
        TX bytes:1681 acl:0 sco:0 commands:39 errors:0
```

3. You will see a list of devices with their name and MAC address.
4. Figure out the mac of your device by turning it off and on and finding the difference.
5. Now to get more information about the device. Do a**“sudo hcitool leinfo –random <mac>”** –random depends on the type addressing.

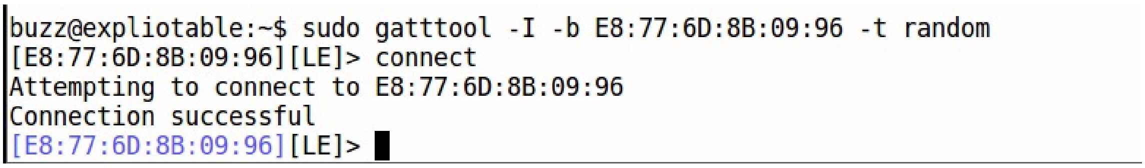


```
buzz@expliotable:~$ sudo hcitool leinfo --random E8:77:6D:8B:09:96
Requesting information ...
        Handle: 70 (0x0046)
        LMP Version: 4.1 (0x7) LMP Subversion: 0x64
        Manufacturer: Nordic Semiconductor ASA (89)
        Features: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

6. You will get basic information like the manufacturer of radio.

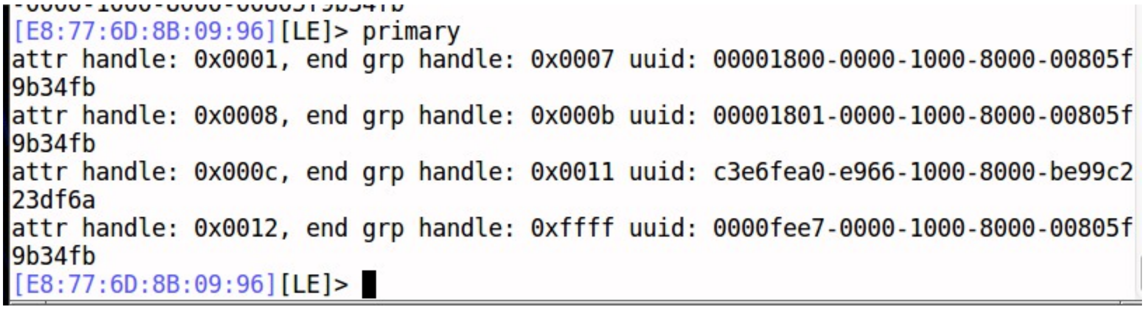
2.6 Reading and writing data

1. Once you got the MAC address of your device. Save it in a file. It will be useful.
2. To connect to a smart device's GATT server. We use a tool called as gatttool.
3. Using this command **"sudo gatttool -I -b <mac> -t random"** you will get a CLI like this and type " connect" to it.

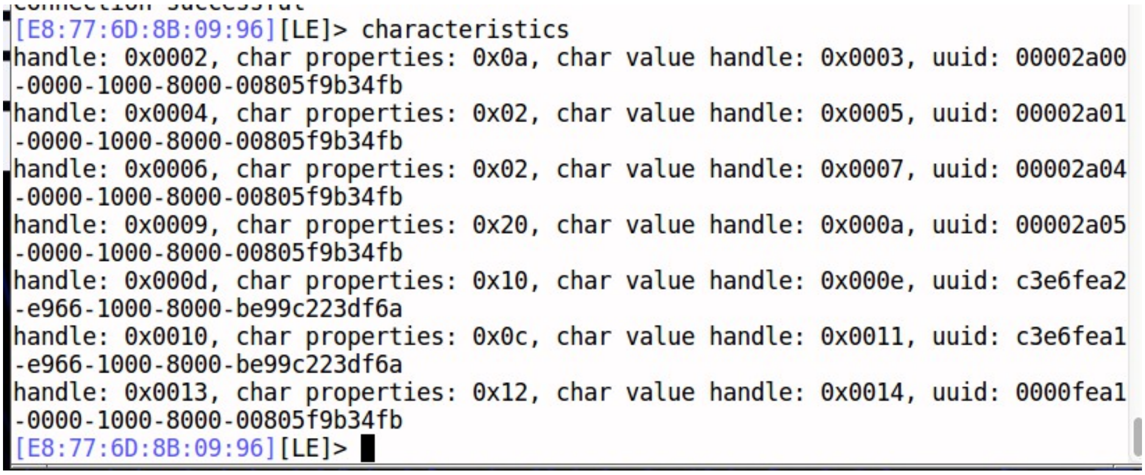


```
buzz@exploitable:~$ sudo gatttool -I -b E8:77:6D:8B:09:96 -t random
[E8:77:6D:8B:09:96][LE]> connect
Attempting to connect to E8:77:6D:8B:09:96
Connection successful
[E8:77:6D:8B:09:96][LE]> 
```

4. Now you can see the characteristics and services running on the device by using **"primary"** , **"characteristics"** and **"char-desc"** to see all the UUIDs running in the device

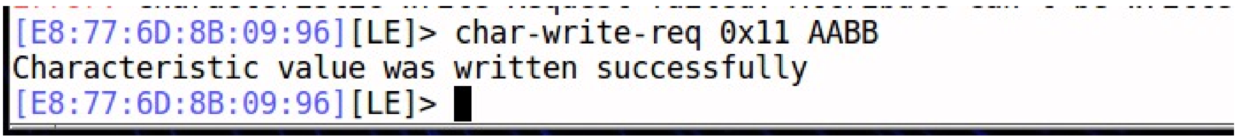


```
[E8:77:6D:8B:09:96][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-1000-8000-00805f
9b34fb
attr handle: 0x0008, end grp handle: 0x000b uuid: 00001801-0000-1000-8000-00805f
9b34fb
attr handle: 0x000c, end grp handle: 0x0011 uuid: c3e6fea0-e966-1000-8000-be99c2
23df6a
attr handle: 0x0012, end grp handle: 0xffff uuid: 0000fee7-0000-1000-8000-00805f
9b34fb
[E8:77:6D:8B:09:96][LE]> 
```

5. 

```
[E8:77:6D:8B:09:96][LE]> characteristics
handle: 0x0002, char properties: 0x0a, char value handle: 0x0003, uuid: 00002a00
-0000-1000-8000-00805f9b34fb
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01
-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x02, char value handle: 0x0007, uuid: 00002a04
-0000-1000-8000-00805f9b34fb
handle: 0x0009, char properties: 0x20, char value handle: 0x000a, uuid: 00002a05
-0000-1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x10, char value handle: 0x000e, uuid: c3e6fea2
-e966-1000-8000-be99c223df6a
handle: 0x0010, char properties: 0x0c, char value handle: 0x0011, uuid: c3e6fea1
-e966-1000-8000-be99c223df6a
handle: 0x0013, char properties: 0x12, char value handle: 0x0014, uuid: 0000fea1
-0000-1000-8000-00805f9b34fb
[E8:77:6D:8B:09:96][LE]> 
```

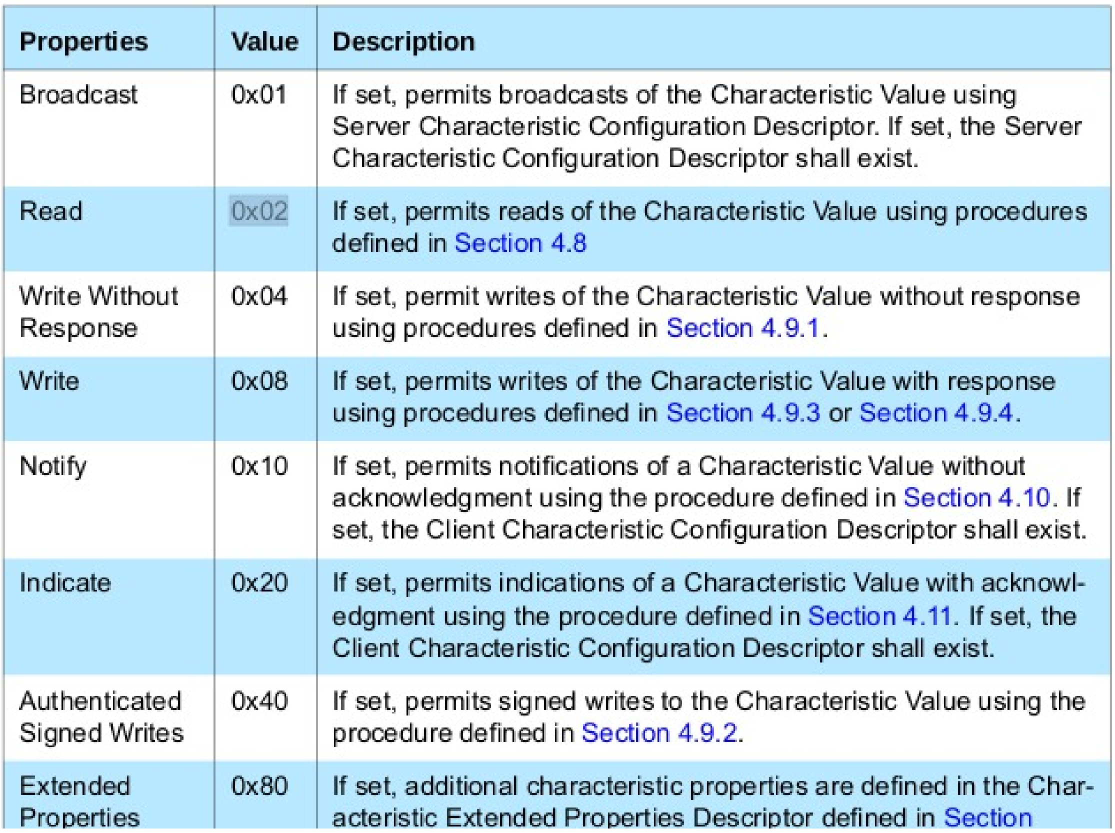
6. Now you can read and write to these handles using "**char-read-hnd <handle>**" and **"charwrite-req <handle> <data>"** to read and write to it.



```
[E8:77:6D:8B:09:96][LE]> char-write-req 0x11 AABB
Characteristic value was written successfully
[E8:77:6D:8B:09:96][LE]> █
```

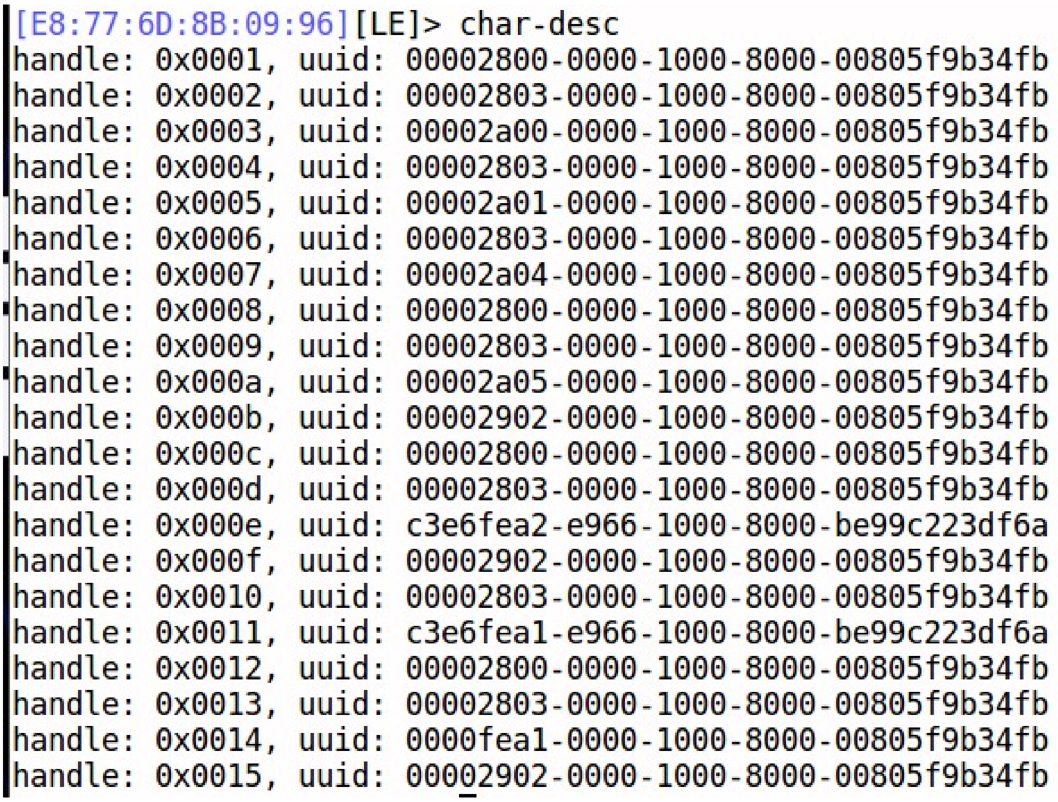
7. Here the char properties give you the permission of the handle like Read, Write, Notify, Indicate.

8. characteristic, as defined by higher layer specifications, without regard to security requirements.



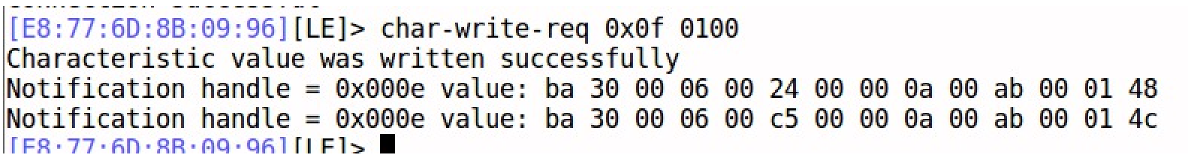
Properties	Value	Description
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value using procedures defined in Section 4.8
Write Without Response	0x04	If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1.
Write	0x08	If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3 or Section 4.9.4.
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgment using the procedure defined in Section 4.10. If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgment using the procedure defined in Section 4.11. If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2.
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section

9



```
[E8:77:6D:8B:09:96][LE]> char-desc
handle: 0x0001, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0002, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0004, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0006, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0007, uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x0008, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0009, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x000a, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x000b, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x000c, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x000d, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x000e, uuid: c3e6fea2-e966-1000-8000-be99c223df6a
handle: 0x000f, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x0010, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0011, uuid: c3e6fea1-e966-1000-8000-be99c223df6a
handle: 0x0012, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0013, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0014, uuid: 0000fea1-0000-1000-8000-00805f9b34fb
handle: 0x0015, uuid: 00002902-0000-1000-8000-00805f9b34fb
```

10 You can enable notification by writing "**01**" to the handle too



```
[E8:77:6D:8B:09:96][LE]> char-write-req 0x0f 0100
Characteristic value was written successfully
Notification handle = 0x000e value: ba 30 00 06 00 24 00 00 0a 00 ab 00 01 48
Notification handle = 0x000e value: ba 30 00 06 00 c5 00 00 0a 00 ab 00 01 4c
[E8:77:6D:8B:09:96][LE]> █
```

ZIGBEE
Chapter 8: ZigBee Protocol – 101

In this chapter, we are going to discuss ZigBee specification and ZigBee protocol architecture in detail. The next chapter will cover the security architecture of ZigBee protocol and security issues present in ZigBee devices and networks.

1. History

ZigBee – developed as an ad-hoc digital radio network during the 1990s and alternate to the wired network. Unlike Wi-Fi or Bluetooth, ZigBee is more suitable for applications in industrial automation and control systems (IACS) where end nodes are required to transmit a low amount of data periodically and be low on power consumption and have a short distance of transmission from a centralized monitoring system.

Nowadays, ZigBee Low-Rate, Wireless Personal Area Network (LR-WPAN) is widely used in control and monitoring applications that require a low data rate, long battery life, self-healing, and secure network in noisy RF environments.

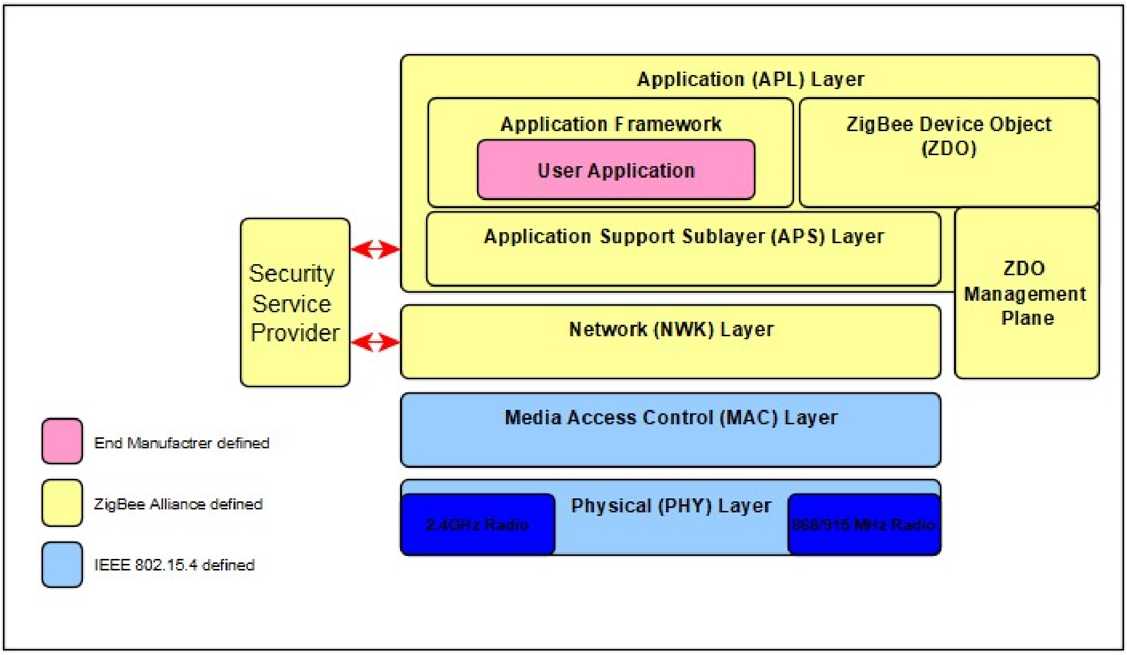
Below are some industries that use ZigBee standard for networking solution in their products

- Wireless sensor networks (WSNs)
- Industrial Automation
- Building Automation
- Home Automation
- Hospitals and Health care Automation
- Smart Energy metering

2. ZigBee Overview

In 2000, IEEE 802.15 working group was formed to work on or standard for wireless personal area networks (WPAN) characterized by a short-range, high level of simplicity, allowing for low cost and low power implementations.

The First edition of IEEE 802.15.4 standards was released in 2003, i.e., IEEE 802.15.4-2003 (LRWPAN), and defined the physical layer and data-link layer of the OSI model. In 2002 ZigBee Alliance was established and, working together with IEEE 802.15.4 working group, announced ZigBee v.1.0 draft ratified ZigBee 2004 Specification in December 2004. ZigBee standard is built on top of IEEE 802.15.4-standards, where IEEE 802.15.4 defines the first two layers, i.e., Physical (PHY) layer, Medium access control (MAC) layer for low-rate wireless personal area network (LR-WPAN), and ZigBee standard provides Network layer (NWK), Application layer (APL) and security features – as shown in the figure below


Application (APL) Layer
Application Framework
User Application
ZigBee Device Object (ZDO)
Application Support Sublayer (APS) Layer
ZDO Management Plane
Security Service Provider
Network (NWK) Layer
Media Access Control (MAC) Layer
Physical (PHY) Layer
2.4GHz Radio
868/915 MHz Radio
End Manufactrer defined
ZigBee Alliance defined
IEEE 802.15.4 defined

Following sections will illustrate the different components of ZigBee standard in more details

3. IEEE 802.15.4 Protocol

3.1 PHY Layer

IEEE 802.15.4 PHY layer provides modulation-demodulation of data, transmission, and reception management services and operates in two separate frequency ranges, low frequency 868/915 MHz and high frequency 2.4 GHz. PHY layer is responsible for Radio controls (enable/disable), Link Quality Indication of received packets (LQI), Energy Detection (ED), and Clear Channel Assessment (CCA)

PHY	Frequency	Channels	Regions
Low Frequency PHY	868.0-868.6 MHZ	1	Europe
Low Frequency PHY	902-928 MHz	30	United States and Australia
High Frequency PHY	400-2483.5 MHZ	16	Worldwide

Source :
ZigBee Specification Document 05-3474-21 (docs-05-3474-21-0csg-zigbee-specification)

3.2 MAC Layer

IEEE 802.15.4 MAC layer controls access to the radio channel using a CSMA-CA mechanism. MAC layer is responsible for Beacon transmission (when the device is a coordinator), implementation of carrier sense multiple access with collision avoidance (CSMA-CA), synchronization using a guaranteed time slot (GTS) mechanism, provide a reliable transmission mechanism for upper layers.

4. IEEE 802.15.4 Network Model

4.1 Node Type

IEEE 802.15.4 standard defines two types of network nodes

4.2 Full-Function Device (FFD)

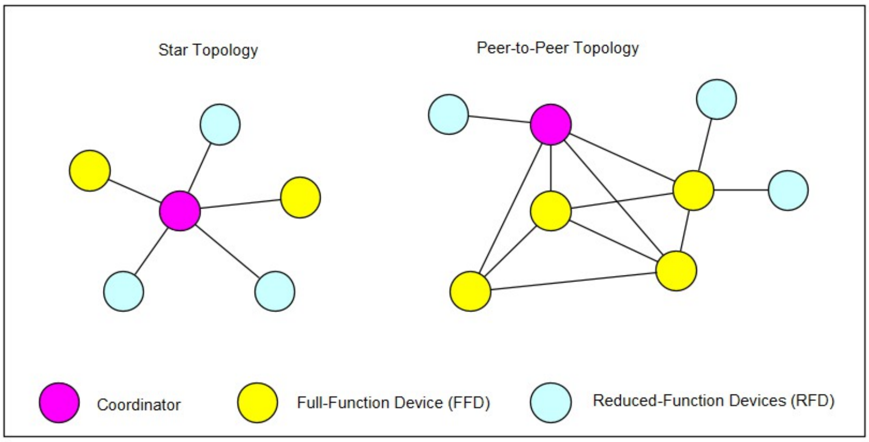
FFD device is capable of network creation, configuration, and message routing within the PAN network. FFD device is capable of configuring the security model in the PAN network. FFD devices can operate in three operating modes, namely: PAN Coordinator, a Coordinator, and End Device. An FFD device can communicate with any RFD or FFD device in the network.

4.3. Reduced-Function Devices (RFD)

RFD is often a battery-operated, simple device with very modest resource and communication requirements. RFD devices can only act as an End Device in the PAN network due to a lack of routing capability and can only communicate with FFD devices in the network.

4.4 Topologies

IEEE 802.15.4 standard defines two network topologies star or peer-to-peer for LR-WPAN devices. However, every network needs at least one FFD to work as the coordinator of the network.



Source :[IEEE 802.15.4 Specification Document (IEEE Std 802.15.4-2020 IEEE Standard for Low-Rate Wireless Networks)]

4.5 Star

The star network is a more structured network with at least one FFD device in the network act as a coordinator for the entire network. Any FFD device chooses its unique PAN identifier to create its own PAN and declares itself as a PAN coordinator. Once the PAN coordinator becomes active in the network, other end devices can join the network. The end device can only communicate to the central node or coordinator. It mostly used in Home-Automation, personal health monitoring, toys, and game control.

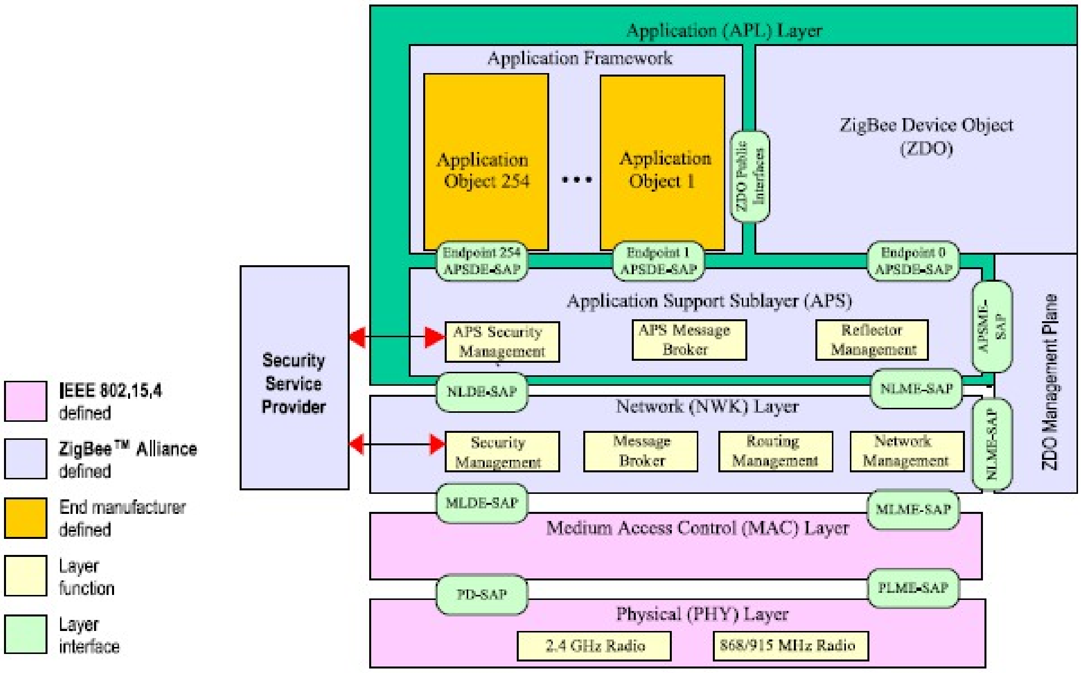
4.6 Peer–to–Peer

The Peer-to-Peer networks also need one PAN coordinator, but unlike a star network, any device can communicate with any other device within the network range. A peer-to-peer network can be an ad-hoc network capable of self-organization and self-management. It is mostly used in industrial control and monitoring systems, wireless sensor networks, inventory management systems.

4.7 Node Addressing Modes

Each IEEE 802.15.4 has two addressing modes, short (16-bit), and extended (64-bit) addressing. An IEEE 802.15.4 compliant device gets its 64-bit extended address from the manufacture while a unique 16-bit PAN address assigned by the coordinator when the device associates with a WPAN.

5. ZigBee Protocol


Application (APL) Layer
Application Framework
Application Object 254
•••
Application Object 1
ZDO Public Interfaces
ZigBee Device Object (ZDO)
Endpoint 254 APSDE-SAP
Endpoint 1 APSDE-SAP
Endpoint 0 APSDE-SAP
Application Support Sublayer (APS)
APS Security Management
APS Message Broker
Reflector Management
APSME-SAP
ZDO Management Plane
Security Service Provider
NLDE-SAP
NLME-SAP
Network (NWK) Layer
Security Management
Message Broker
Routing Management
Network Management
NLME-SAP
MLDE-SAP
MLME-SAP
Medium Access Control (MAC) Layer
PD-SAP
PLME-SAP
Physical (PHY) Layer
2.4 GHz Radio
868/915 MHz Radio
IEEE 802.15.4 defined
ZigBee™ Alliance defined
End manufacturer defined
Layer function
Layer interface

Source: ZigBee Specification Document 05-3474-21 (docs-05-3474-21-0csg-zigbee-specification)

5.1 Application Layer

As shown in the figure above, the APL layer consists of several sublayers, namely: APS sublayer, service access points (SAP), and the ZigBee Device Object (ZDO), along with the ZDO management plane and the manufacturer defined application objects.

5.2 Application Support Sub-Layer (APS)

The APS sub-layer provides an interface between the NWK and APL through a general set of services that are used by both the ZDO and the manufacturer-defined application objects. The APS Data Entity (APS-DE) and APS Management Entity (APS-ME) are two APS entities and provide the below services.

5.3 APS data entity (APS-DE)

- Generation of the Application support sub-layer protocol data unit (APDU)
- Device Binding
- Group address filtering
- Reliable transport
- Duplicate rejection
- Fragmentation

5.4 APS management entity (APM-SE)

- Binding management
- Application support layer information base (AIB) management
- Security
- Group management

5.5 Application Framework

Application Framework provides an execution environment in which application objects are hosted and can send or receive data, up to 254 distinct application objects can be defined, each identified by an endpoint address from 1 to 254.
Endpoint 0 and Endpoint 255 are used as ZigBee Device Object (ZDO) address and broadcast address respectively by Application support sub-layer data entity - service access point (APSDESAP) Endpoints 241 through 254 are reserved by ZigBee Alliance and cannot be used without approval.

5.6 Application Profiles

Application profiles are agreements for messages, message formats, and processing actions that enable developers to create an interoperable, distributed application employing application entities that reside on separate devices.

ZigBee Alliance has published a number of public application profiles for applications like Home Automation, Industrial Automation, etc. Device manufactures can also define custom profiles suitable for their end application.

5.7 Clusters

Clusters are represented as the collection of attributes and application messages. Clusters are divided into two types, i.e., input and output clusters.

A cluster identifier is a 16-bit number unique within the scope of a particular application profile.

5.8 ZigBee Device Objects (ZDO)

Located between the application framework and APS, the ZigBee device objects (ZDO) represent a base class of functionality that provides an interface between the application objects, the device profile, and the APS.

The ZDO is responsible for the following:

- Initializing the application support sub-layer (APS), the network layer (NWK), and the Security Services.
- Assembling configuration information from the end applications to determine and implement discovery, security management (key loading, key establishment, key transport, and authentication), network management (network discovery, leaving/joining a network, resetting a network connection and creating a network), and binding, node, and group management.

6. Network Layer

The network layer provides a service interface between the 802.15.4 MAC layer and the application layer.

The Network Layer Data Entity (NLDE) and Network Layer Management Entity (NLME) are two network layer entities provides below services:

6.1 Network Layer Data Entity (NLDE)

- Generation of the Network level PDU (NPDU)
- Topology-specific routing
- Security

6.2 Network Layer Management Entity (NLME)

- Configuring a new device
- Starting a network
- Joining, rejoining and leaving a network
- Addressing:
- Neighbor discovery
- Route discovery
- Reception control
- Routing

7. ZigBee Device Type

A ZigBee device can work in three different modes or node types. Node types in ZigBee network are mentioned below

7.1 ZigBee Coordinator (ZC)

A ZigBee coordinator is an FFD device that acts as a central node or parent for other nodes in the network. Only one per network is responsible for the creation, configuration, and management of the ZigBee network. It maintains a list of associated devices, support services like association, disassociation, and orphan scan, rejoin. ZigBee network cannot exist without a ZC; ZC is always active on the network and cannot be put in sleep mode.

7.2 ZigBee Router (ZR)

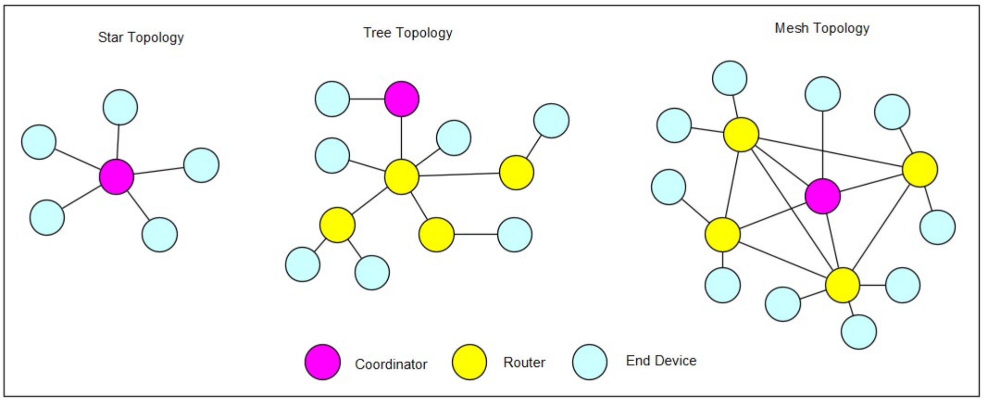
The ZigBee Router is an intermediate FFD device, responsible for relaying packets between end devices or between an end device and the coordinator. End Device can also join the network through a router, with the ZR acting as a parent for that segment of the network.

7.3 ZigBee End Device (ZED)

Any FFD or RFD device can become an End device on a ZigBee network, ZigBee End device is a simple device like a sensor responsible for monitoring and collecting data, or take particular action based commands from the user. ZED, without any message routing capability, can only send and receive data from the parent node. Usually, ZED is are low powered battery operated device and can be put to sleep to save power consumption.

8. Network Topology

The ZigBee network layer (NWK) supports star, tree, and mesh topologies.



Source :
[ZigBee Specification Document 05-3474-21 (docs-05-3474-21-0csg-zigbee-specification)]

8.1 Star

A Single ZigBee coordinator device per network, controlling the network, and is responsible for initiating and maintaining the ZigBee network. All other devices, called end devices, directly communicate with the ZigBee coordinator or other end devices via ZigBee coordinator.

In the Star network, the coordinator becomes a bottleneck for message routing, and the failure of the coordinator leads to network shutdown.

8.2 Tree

In tree topologies, the ZigBee coordinator is responsible for starting the network and for choosing specific key network parameters, but the network may extend by ZigBee routers. In tree networks, routers route data and control messages through the network using a tree routing strategy.

In a Tree network, the failure of a router can lead to a shutdown of the network segment under the affected router.

8.3 Mesh

In a mesh topology, the ZigBee coordinator is responsible for creation and configuration, but the network may extend using ZigBee routers. Mesh networks allow full peer-to-peer communication. Mesh network, also called a self-healing network, i.e., failure of coordinator, does not result in network failure as end devices communicate with each other and to the router. Mesh network is complex and creates a messaging overhead in the network.

9. Addressing in ZigBeey

9.1 Device Addressing

Each ZigBee device in a ZigBee network gets two types of addresses - an IEEE address and a Network Address

9.2 IEEE Address

A globally unique 64-bit address and assigned by device manufacture during production. Part of IEEE 802.15.4 standard, this address is also referred to as “extended address” used by IEEE 802.15.4 layer for low-level packet delivery and rarely used by the ZigBee layer except for the binding map need to create a binding between IEEE address and Network Address.

9.3 Network Address

A unique within a ZigBee network, 16-bit Network address is part of the ZigBee layer, also called the “short address”. Network Address is assigned to the end device and used by the network layer for routing messages between devices.

9.4 ZigBee Network Identity

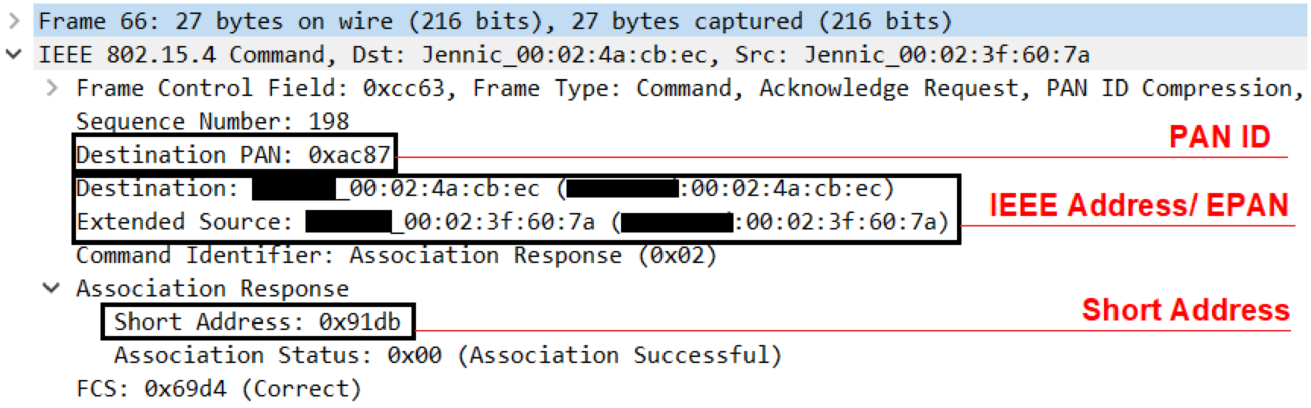
Each ZigBee device in a ZigBee network gets two types of identifiers (ID) - the Personal Area Network Identifier (PAN ID) and the Extended PAN ID (EPID).

9.5 PAN Identifier (PAN ID)

Part of the IEEE 802.15.4 standard, the PAN identifier (PAN ID) is a 16-bit identifier selected by the PAN Coordinator while setting up the network and communicated to the End devices. MAC layer filter out packets from other networks which are not part of the same network as identified by the specific PANID.

9.6 Extended PAN ID (EPID)

A globally unique 64-bit identifier of the ZigBee network, EPID identifier should be unique among the PAN overlapping in a given area and is used to avoid PAN ID conflicts between distinct networks. Refer below to a snapshot of ZigBee packet for more understating of PAN ID, EPAN, and a short address



```
> Frame 66: 27 bytes on wire (216 bits), 27 bytes captured (216 bits)
v IEEE 802.15.4 Command, Dst: Jennic_00:02:4a:cb:ec, Src: Jennic_00:02:3f:60:7a
  > Frame Control Field: 0xcc63, Frame Type: Command, Acknowledge Request, PAN ID Compression,
    Sequence Number: 198
    Destination PAN: 0xac87
    Destination: [redacted]_00:02:4a:cb:ec ([redacted]:00:02:4a:cb:ec)
    Extended Source: [redacted]_00:02:3f:60:7a ([redacted]:00:02:3f:60:7a)
    Command Identifier: Association Response (0x02)
  v Association Response
      Short Address: 0x91db
      Association Status: 0x00 (Association Successful)
    FCS: 0x69d4 (Correct)
```

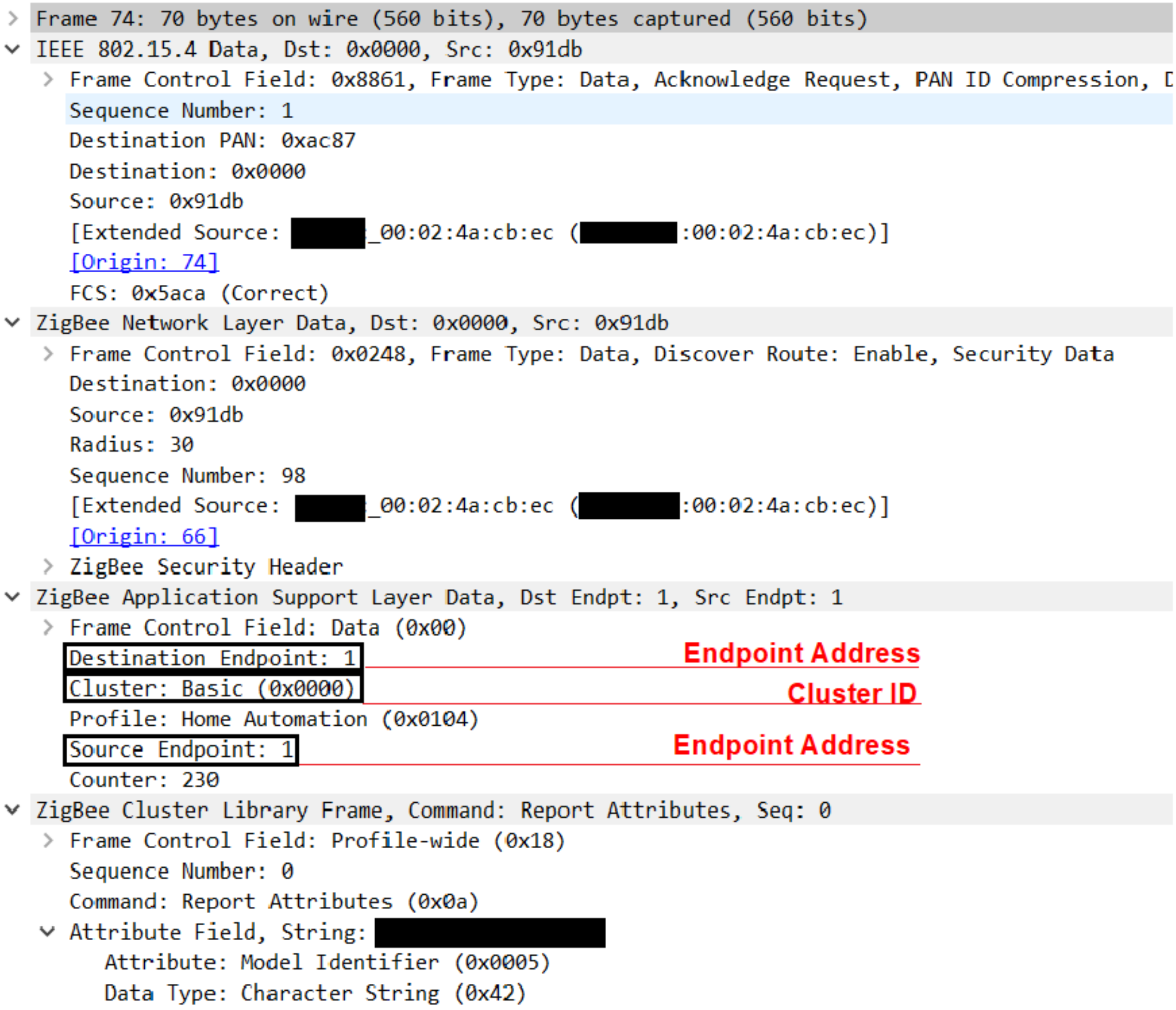
9.7 Application level addressing

9.7.1 End Point Address

Every ZigBee node may support one or more applications or End Points. To send direct messages to a specific application, Endpoints numbered from 1 to 240 are used - there's also a broadcast Endpoint, 255, which allows messages to send to all applications on a given node.

9.7.2 Cluster Identifier (Cluster ID)

The cluster identifier is a 16-bit number unique within the scope of each application profile and identifies a specific cluster. Refer below a snapshot of a ZigBee packet for more understating of End Point address and Cluster ID



```
> Frame 74: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
v IEEE 802.15.4 Data, Dst: 0x0000, Src: 0x91db
  > Frame Control Field: 0x8861, Frame Type: Data, Acknowledge Request, PAN ID Compression,
    Sequence Number: 1
    Destination PAN: 0xac87
    Destination: 0x0000
    Source: 0x91db
    [Extended Source: [redacted]_00:02:4a:cb:ec ([redacted]:00:02:4a:cb:ec)]
    [Origin: 74]
    FCS: 0x5aca (Correct)
v ZigBee Network Layer Data, Dst: 0x0000, Src: 0x91db
  > Frame Control Field: 0x0248, Frame Type: Data, Discover Route: Enable, Security Data
    Destination: 0x0000
    Source: 0x91db
    Radius: 30
    Sequence Number: 98
    [Extended Source: [redacted]_00:02:4a:cb:ec ([redacted]:00:02:4a:cb:ec)]
    [Origin: 66]
  > ZigBee Security Header
v ZigBee Application Support Layer Data, Dst Endpt: 1, Src Endpt: 1
  > Frame Control Field: Data (0x00)
    Destination Endpoint: 1
    Cluster: Basic (0x0000)
    Profile: Home Automation (0x0104)
    Source Endpoint: 1
    Counter: 230
v ZigBee Cluster Library Frame, Command: Report Attributes, Seq: 0
  > Frame Control Field: Profile-wide (0x18)
    Sequence Number: 0
    Command: Report Attributes (0x0a)
  v Attribute Field, String: [redacted]
      Attribute: Model Identifier (0x0005)
      Data Type: Character String (0x42)
```

10. Messaging in ZigBee

ZigBee supports broadcast, unicast, group-multicast, and Inter-PAN messaging.

10.1 Broadcast

Message initiated by PAN coordinator, intended for all devices in the PAN network and router devices, the router retransmits it for End Node belonging to the same PAN network. Below table indicates the broadcast address used

Sr. No.	Broadcast Address	Destination Group
1	0xFFFF	All devices in PAN
2	0xFFFE	Reserved
3	0xFFFD	Devices with receiver turned on permanently, macRxOnWhenIdle = TRUE
4	0xFFFC	All routers and coordinator
5	0xFFFB	Low power routers only
6	0xFFF8 - 0xFFFA	Reserved

10.2 Unicast

Message directed towards a single End Node in the network and often routed through multiple nodes to reach the destination node. A unicast message requires a network-level acknowledgment from the final destination device to the source device, while MAC level acknowledgment is sent between the MAC layer if the message route through different nodes.

10.3 Group Multicast

Message intended for every device in a particular PAN belonging to a dynamically defined multicast group, and within a given transmission radius measured in hops.

10.4 Inter-PAN Communication

Messages intended for a device in a different PAN network with a different PAN ID. ZigBee specification defines inter-pan communication as a mechanism whereby ZigBee devices can perform exchanges of information with devices in their local area without having to form or join the same ZigBee network.

11. Conclusion

We hope this gives you an introduction and a brief insight into the ZigBee protocol. In the next chapter, we will look at the security of the ZigBee protocol and devices.

Chapter 9: ZigBee Security - 101

In this chapter, we will discuss the security architecture of ZigBee protocol and security issues present in ZigBee devices and networks.

1. ZigBee Security Architecture (Open Trust)

The security architecture in ZigBee complements or enhances the security service of the IEEE 802.15.4 layers. It is an "open trust" model based on certain assumptions, which are described below:

- Different layers and applications are running on a single device trust each other.
- The communication between different stack layers on the same device is not encrypted.
- A device will not intentionally or inadvertently transmit keys to other devices unless protected, such as during key-transport.
- The communication between the two devices is cryptographically encrypted and secured.
- Random number generators are working as expected by the cryptographic engine
- Hardware is tamper-resistant

ZigBee security architectural design principle:

- The layer that originates a frame is responsible for initially securing it.
- Only a device with an active network key can communicate to more than one hop across the network.
- Both the APS layer and NWK layer can use the same active network key to secure the frames. Re-use of keys helps reduces storage overhead.
- End to End message security, i.e., the only source and destination devices, can decrypt the messages protected by a shared key, and the routing mechanism is out of trust considerations.
- A device that forms a network is responsible for base security level, security policies, and authentication of nodes in the network. The application layer can provide additional application-level security if required between two devices.

2. Trust Center

The Trust Center is an application that runs on a device trusted by other devices within a ZigBee network to distribute keys for network and potentially end-to-end application configuration management. Only one trust center can exist per network, and it can be a coordinator or a device designated by the coordinator, and all member nodes recognize

this device as a trust center. Trust center is responsible for
- Configuring and maintaining network security policies
- Establishing end-to-end application keys.
- Generation of keys by using some key establishment protocol

3. Security Modes in ZigBee

3.1 Distributed Security Mode

The distributed Security Mode, a unique Trust Center, is not required in the network, and routers are responsible for end device authentication. The link key is pre-configured on the device, and the network key is issued by a router when the device joins the network. Network key remains the same for all nodes in the network; this makes distributed security mode less secure.

3.2 Centralized Security Mode

The centralized security mode used in applications, a trust center control, and maintain centralized security policy for network and device. In this mode, the trust center is responsible for
- aintaining security and security configuration for the entire network.
- Authentication of devices and maintaining a list of devices on the network.
- Maintaining Link keys and Network keys with all the devices in the network.

4. ZigBee Security Keys

ZigBee standard defines two types of symmetric keys, each of 128-bit length used for encrypted communication.

4.1 Network Key

128-bit Network key used in broadcast communication and any network layer communications.
Each node requires the network key to communicate securely with other devices on the network. A device on the network acquires a network key via key transfer on the network, i.e., keytransport.

There is only one type of network key; however, it can use in either distributed or centralized security models. The security model controls how a network key is distributed and may control how network frame counters initialized. The security model does not affect how messages are secured.

4.2 Link Key

A 128-bit unique Link key shared by two devices, used in unicast communication between APL peer entities. A device can get link keys either via key-transport service over the network, or preinstallation

There are two different types of trust center link keys: global and unique.
The type of trust center link key in use by the local device determines how the device handles various trust center messages (APS commands), including whether to apply APS encryption or not.

Each node may also have the following pre-configured link keys, which would use to derive a Trust Center link key.

Sr. No.	Key Name	Description
1	Centralized security global trust center link key	Link key used for joining centralized security networks. The default value for the centralized security global trust center link key is ZigBeeAlliance09, i.e. ("5A 69 67 42 99 23 65 65 41 6C 6C 69 61 6E 63 65 30 39") and is used or supported by the device if no other link key is specified by the application at the time of joining
2	Distributed security global link key	Link key used for joining distributed security networks
3	Install code link key	Link key derived from install code from joining device to create unique trust center link key for joining
4	Application link key	Link key used between two devices for application -layer encryption
5	Device Specific trust center link key	Link key used between the trust center and a device in the network. Used for trust center commands and application-layer encryption.

Source : ZigBee Specification Document 05-3474-21 (docs-05-3474-21-0csg-zigbee-specification)

5. Key Management

ZigBee specification mentions three different key management mechanisms

- Pre-installation – Keys are configured or install on devices out-of-band
- Key Transport – The Trust center sends security keys over the network to the device.
- Key Establishment – Trust center and end device negotiate on keys, and keys are established without ever actually sending any key over the network. This key establishment is based on the Symmetric-Key Key Establishment (SKKE) protocol.

6. ZigBee Protocol Security

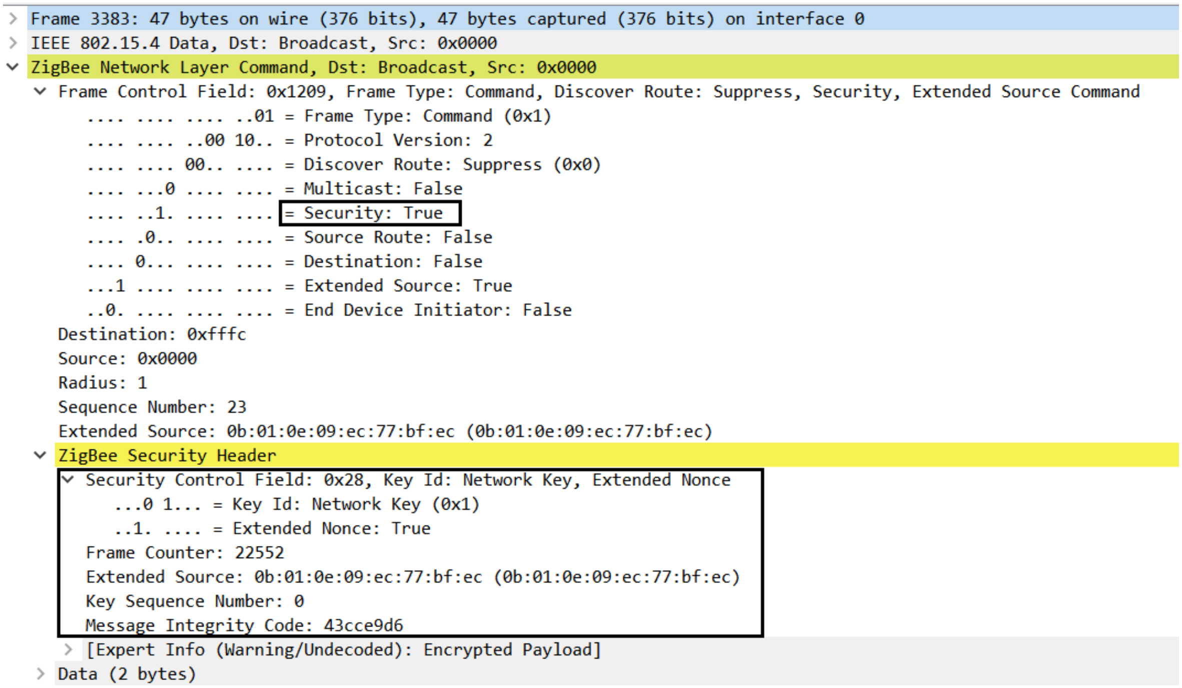
6.1 Auxiliary Security Header

ZigBee NWK layer and APS layer can use the Auxiliary Security Header for secure communication if security control bit in the NWK frame control field set to 1.
Auxiliary Security Header has the following fields

Octets: 1	Octets: 4	Octets: 0 or 8	Octets: 0 or 1
Security Control	Frame Counter	Source Address	Key Sequence

Source :
ZigBee Specification Document 05-3474-21 (docs-05-3474-21-0csg-zigbee-specification)

Below is a sample snapshot of ZigBee packet with security field in NWK frame control set to true (1) and Security header is set with corresponding security control fields, and Message Integrity Code (MIC)



```
> Frame 3383: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface 0
> IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0000
v ZigBee Network Layer Command, Dst: Broadcast, Src: 0x0000
  v Frame Control Field: 0x1209, Frame Type: Command, Discover Route: Suppress, Security, Extended Source Command
      .... .... .... ..01 = Frame Type: Command (0x1)
      .... .... ..00 10.. = Protocol Version: 2
      .... .... 00.. .... = Discover Route: Suppress (0x0)
      .... ...0 .... .... = Multicast: False
      .... ..1. .... .... = Security: True
      .... .0.. .... .... = Source Route: False
      .... 0... .... .... = Destination: False
      ...1 .... .... .... = Extended Source: True
      ..0. .... .... .... = End Device Initiator: False
    Destination: 0xfffc
    Source: 0x0000
    Radius: 1
    Sequence Number: 23
    Extended Source: 0b:01:0e:09:ec:77:bf:ec (0b:01:0e:09:ec:77:bf:ec)
  v ZigBee Security Header
    v Security Control Field: 0x28, Key Id: Network Key, Extended Nonce
        ...0 1... = Key Id: Network Key (0x1)
        ..1. .... = Extended Nonce: True
      Frame Counter: 22552
      Extended Source: 0b:01:0e:09:ec:77:bf:ec (0b:01:0e:09:ec:77:bf:ec)
      Key Sequence Number: 0
      Message Integrity Code: 43cce9d6
    > [Expert Info (Warning/Undecoded): Encrypted Payload]
  > Data (2 bytes)
```

6.2 Security control

8 Bit security control field consists of a security level, a key identifier, and an extended nonce subfield as shown below

Bit	0 - 2	3 -4	5	6 - 7
Description	Security Level	Key Identifier	Extended nonce	Reserved

- Security Level

The security level identifier indicates how an outgoing frame is to be secured, how an incoming frame purportedly has been secured. It also indicates whether or not the payload is encrypted and to what extent data authenticity over the frame provided, as reflected by the length of the message integrity code (MIC). The bit-length of the MIC may take the values 0, 32, 64, or 128 and determines the probability that a random guess of the MIC would be correct. The security properties of the security levels listed in below

Security Level Identifier	Security Attributes	Data Encryption	Frame Integrity (length M of MIC, in Number of Octets)
0x00	None	OFF	NO (M=0)
0x01	MIC-32	OFF	Yes (M=4)
0x02	MIC-64	OFF	Yes (M=8)
0x03	MIC-128	OFF	Yes (M=16)
0x04	ENC ON	NO	(M=0)
0x05	ENC-MIC-32	ON	Yes (M=4)
0x06	ENC-MIC-64	ON	Yes (M=8)
0x07	ENC-MIC-128	ON	Yes (M=16)

Source : ZigBee Specification Document 05-3474-21 (docs-05-3474-21-0csg-zigbee-specification)

- Key Identifier

The key identifier sub-field consists of two bits used to identify the key used to encrypt the frame.

Key Identifier	Description
0x00	A data key
0x01	A network key
0x02	A key-transport key
0x03	A Key-load key

- Extended nonce

When set to 1, the extended nonce sub-field indicates the sender address field is present in the auxiliary header. Otherwise, is set to 0.

6.3 Frame counter

The counter field is used to provide frame freshness and to prevent the processing of duplicate frames.

6.4 Source address

The source address field in security control is the extended 64-bit address of the source device and present when the extended nonce sub-filed of the security control field is set to 1.

6.5 Key sequence number

The key sequence number present in the auxiliary security header indicates the key sequence number of the network key used to secure the frame. The key identifier subfield from the security control field, when set to 1 (i.e., a network key), indicates a key sequence number present in the auxiliary security header.

ZigBee Vulnerabilities

> **Security between devices depends on secure initialization and installation of security keys**

Vulnerabilities in the ZigBee network are categorized into two categories as protocol vulnerabilities and poor implementation of protocol during product development. Below are some common vulnerabilities:

7. Implementation vulnerabilities

7.1 Security keys stored insecurely

ZigBee protocol expects all security keys (network, Link) stored secularly on the device. Keys can identify by reverse-engineering the firmware binary to find the location of the keys if they are not stored securely.

7.2 Over-The-Air insecure key transportation

In some implementations, when a node joins a ZigBee network for the first time, the node obtains its keys over-the-air, mostly in a clear-text format from the coordinator. Thus a sniffer device network sniper or rough device can obtain keys from the coordinator and can compromise the entire network.

7.3 Energy Depletion Attack

Below are two very common energy depletion attacks.

7.4 Invalid security header

In such attacks, an attacker sends bursts of packets with invalid security headers in frame with the intention that the device has to spend some amount of energy to verify frame integrity, leading to faster battery depletion of the target device.

7.5 Polling Rate

In such attacks, attackers send packets to the end device faster than the configured polling rate of the network, leading to faster battery depletion of the target device.

8. Protocol vulnerabilities

8.1 Network Jamming Vulnerability

IEEE802.15.4/ZigBee standards provide certain protection mechanism against radio and network interference, but there are certain techniques by which an attacker can jam the network.

Below are two types of jamming attacks possible in the ZigBee network:

8.1.1 Radio Jamming

In such attacks, the attacker increases the radio signal's emission for a given channel, leading to a decrease in Signal to Noise Ratio of the radio channel.

8.1.2 Link Layer Jamming

In such attacks, the attacker targets the MAC layer by transmitting bursts of random ZigBee frames with useless data on the network either at the random interval or specific interval targeting specific node and thus leading to packet drop and DoS attack in the network.

8.1.3 Link key vulnerability

ZigBee standard has an open-trust model for security and below vulnerabilities in standard leads to Link key related attack

8.1.4 Default Link Key

ZigBee standard provides a default value for link key to ensure interoperability between ZigBee devices from different manufacturers; thus, an attacker can use a rogue device to join the network with the default network key.

8.1.5 Unencrypted Link-Key

When a device without pre-configured key tries to join the network, in such cases trust center sends a single key (default link key) unencrypted to the device and can be obtained by an attacker by sniffing the network communication leading to ZigBee network compromise.

8.1.6 Re-using Link key

ZigBee standard permits the re-use of link keys for rejoining the network; in such cases, an attacker can clone the legitimate device and spoof the network layer of Trust Center by pretending to be previously connected device that wants to rejoin the network. Thus Trust Center then sends the keys encrypted with the previously used link key.

8.2 Unauthenticated ACK frame vulnerability

Acknowledgment frame is a part of the network layer in IEEE 802.15.4/ZigBee standards but limited to confirmation of frame transmission at the network layer and does not provide frame integrity and confidentiality protections for acknowledgment packets. Below are some common ACK attacks in the ZigBee network. Both required Link Layer jamming.

8.2.1 ACK Spoofing

In such attacks, an attackers jam the network such that the legitimate device does not receive frames, and then the attacker sends the ACK frame with the correct sequence number to the sender, leading to data loss in the network.

8.2.2 ACK Dropping

In such attacks, attacker jams the network such that only the ACK frame from receiver to the sender jammed, forcing the sender to retransmit data till the maximum number of retransmissions, depletes the network bandwidth and device battery power.

8.3 Replay-protection vulnerability

An IEEE 802.15.4 has replay-protection mechanisms. It mentions that a node can drop the received frame if the frame sequence number is equal to or less than the sequence number of the preceding frame received from the same source node. In such attacks, an attacker can send frames with large sequence numbers to the target node, forcing the target node to drop the frame with a smaller sequence number.

9. Conclusion

We hope this chapter gave you a brief insight into the security architecture of the ZigBee protocol. The vulnerabilities mentioned in this chapter are either due to a lack of secure design and implementation or due to inherent vulnerabilities of the ZigBee protocol.

Reference :

Chapter 5 : MQTT Broker Security - 101

- Mosquitto broker configuration file
- mosquitto_passwd utility
- Configure SSL/TLS support for Mosquitto
- Security in AWS IoT

Chapter 7 : Bluetooth Low Energy - 101

- https://www.nordicsemi.com/eng/News/ULP-Wireless-Update/A-short-history-of-Bluetooth
- https://www.bluetooth.com/specifications/gatt/generic-attributes-overview
- https://www.bluetooth.com/specifications
- https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt
- https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch01.html
- https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
- https://www.jaredwolff.com/blog/get-started-with-bluetooth-low-energy/
- http://object-network.blogspot.com/2014/01/scanning-ble-adverts-from-linux.html
- https://elinux.org/images/3/32/Doing_Bluetooth_Low_Energy_on_Linux.pdf
- https://www.digikey.com/Web%20Export/Supplier%20Content/Laird_776/PDF/laird-wireless-bluetooth-smart-ready.pdf

Chapter 8 : ZigBee Protocol - 101

- 802.15.4-2020 - IEEE Approved Draft Standard for Low-Rate Wireless Networks
- ZigBee specification - 05-3474-21, August 5, 2015
- The ZigBee Alliance
- IEEE 802.15.4 Wikipedia
- ZigBee Wikipedia

Chapter 9 : ZigBee Security - 101

- 802.15.4-2020 - IEEE Approved Draft Standard for Low-Rate Wireless Networks
- ZigBee specification - 05-3474-21, August 5, 2015
- The ZigBee Alliance
- IEEE 802.15.4 Wikipedia
- ZigBee Wikipedia

Part 3

RADIO

Chapters: 10 to 11

What to expect:

Zero-in on the basics of Software-Defined Radio, the requisite tools, and how to go about approaching an RF target.

Chapter 10: Introduction to Software Defined Radio: Hardware Tools

---

1. Introduction

In layman's terms, Software Defined Radio is the implementation of major signal processing components i.e., modulators/demodulators, encoders/decoders, amplifiers, mixers (that are typically implemented in hardware) within the software. These software platforms are very generic and support all types of frequencies as well as different analyses on them. Even though the hardware is still a major requirement (transceiver and antenna) but the introduction of software-defined radio certainly created a wave in IoT as well as other major areas where wireless communication is used and made signal processing much more accessible than it used to be previously.

Wireless communication is one of the major attack surfaces in an IoT environment. SDR is the technology to build these wireless products, analyze the communication, and, most importantly, break it! Not only does SDR cover all major wireless communication protocols used in IoT i.e. Bluetooth, Zigbee, Wi-Fi, NFC, but it offers a vast range of other possible attacks that can be used to compromise the security of a device or infrastructure. A few of these are listed below:

- Cryptanalysis attack
- Side-channel attack like Tempest
- Cellular or Mobile network attacks
- SAT-comm analysis

2. Prerequisite

> "You can't run away from maths and theory when doing signal processing."

Well, someone said it right :P But you don't have to be a maths geek to get started with Softwaredefined radio, but the knowledge of the basic concepts and terminologies will provide an edge while assessing your target and will help in the long run:

- Basics of analog and digital signal
- Concepts of digital signal processing
- Communication systems
- Complex numbers
- Signals and systems
- Types of modulation
- Types of Encoding

We feel that the following topics are more than sufficient to help you get started with RF communication and assist you while learning about SDRs. We have some useful resources at the end of the chapter that will help you understand these topics better.

3. Tools of trade :

> "A tool is but the extension of a man's hand..." -H.W. Beecher

Even though most of the signal processing is done via software but hardware tools are also required to transmit and receive the signals. Some hardware and software tools that are necessary for SDR are listed below(we will elaborate on a few of the tools mentioned below as we proceed):

Hardware

Basically, when we pick an SDR peripheral device, the things to keep in mind are its:

- Operating frequency range:

This defines the range of frequency the peripheral device is capable of performing operations upon (Tx and Rx). It's always nice to have a wider operating frequency range at your disposal.

- RX Bandwidth:

Greater the RF front end bandwidth (Rx bandwidth), the more our SDR peripheral can gather data around the operating center frequency. Keeping in mind that with wider RF bandwidth, we always gather in more noise as well.

- ADC Resolution (in Bits) :

Usually, an ADC which has a higher resolution is preferred, but keeping in mind with higher ADC resolution, we need to decrease the signal processing overhead such that we can process the samples in real-time. Hence choosing a proper ADC resolution is important.

- Transmitting and receiving capabilities:

While most of the peripheral devices can receive(Rx) but not all can transmit(Tx). So it's safe to say a transceiver(Tx-Rx both are present) is preferred as many attacks require transmitting(Tx) of signals as well. Also, based on the requirement, the transceiver can be categorized under halfduplex, full-duplex, and so on.

- Cost:

This is probably the major factor as most of these peripheral devices cost a lot of money. We need to choose one carefully, that fulfills all our requirements based on assessment type.

Now let's jump into some popular SDR devices present out there:

3.1 Realtek SDR Dongle (RTL820T2)

Ask any SDR hobbyist or professional as to what their first tool was; it'll probably be an RTL-SDR dongle. Although for a cost this low, it provides a range from 24-1766MHz, which is quite impressive. A few can other variants can go up to 2.2GHz. The only setback is that it is an SDR receiver, doesn't have any TX capabilities (not considering leakage from its internal local oscillator). Other popular variants are NooElec R820T, R820T, and Terratec R820T.

RTL-SDR.COM
DVB-T + DAB + FM + SDR
RTL2832U R820T2 TCXO+BIAS T+HF

RTL-SDR R820T2 Dongle
Image source: https://commons.wikimedia.org/wiki/File:Rtl-sdr.jpg

3.2 HackRF One

It is a widely used and recommended tool by hobbyists due to its comparatively reasonable cost and exceptional frequency range, 10MHz to 6GHz (Practical range). It is an SDR transceiver, i.e., It has both transmit (Tx) and receive (Rx) capabilities. So for someone who's looking for a massive frequency range, wide bandwidth, and transmit capability, it's the perfect match.

HackRF One
GREAT SCOTT GADGETS

HackRF One by Great Scott Gadgets
Image source: https://commons.wikimedia.org/wiki/File:Hackrf-one-img_0005.jpg

3.3 BladeRF 2.0 micro

One can't miss out on Blade RF when it comes to a powerful SDR peripheral. Even though it has an operating frequency ranging from 47MHz to 6GHz and 56MHz of bandwidth, what sets it apart is the presence of a powerful FPGA processor, which means it can easily do some of the signal processing itself on board. Making it an ideal standalone device. The reason being there are quite a few interesting projects based on BladeRF that include:

- YateBTS
- ATSC Transmitter
- RAMEAR to name a few.

3.4 USRP B210

Ettus products are mostly used by professionals and Industries due to their high capabilities, making it the most expensive peripheral device compared to the others. It has a very high sample rate and a broad tuning rate. Talking of B210, it operates on freq 70Mhz to 6Ghz, being full-duplex and having the maximum sample rate among the above listed of 61.44MS/s.

Ettus
Research
www.ettus.com
RF 1
RF 2
RF 3
RF 4
CLK I/0
6VDC
USB
USRP
Universal Software Radio Peripheral
TIME CONST.

USRP by Ettus
Image source: https://commons.wikimedia.org/wiki/File:Usrp.jpg

Here's a brief comparison between a few popular SDR peripheral devices:

Name	**Frequency Range**	**Bandwidth**	**ADC Resolution**	**Tx capability**	**Price**
RTL-SDR R820T2/RTL2838U	0.5 – 1766 MHZ	Matches sampling rate,but with filter roll-off	8	NO	USD 24
HackRF One	1 MHz - 6 Ghz	20 MHz	8	YES	USD 299
bladeRF	47 MHz – 6 Ghz	56 MHz	12	YES	USD 480
USRP B210	70 MHz – 6 Ghz	56 MHz	12	YES	USD 1,100
LimeSDR	100 kHz – 3.8 GHz	61.44 MHz	12	YES	USD 299
YARD Stick One	300 MHz – 1GHz	-	8	YES	USD 100

4 Antennas

While the selection of an antenna following things are kept in mind:

- Antenna Gain:
  As most of the gains in the antenna datasheets are referred to an isotropic antenna, it is necessary to see where the 3dB gain and bandwidth are located and do some basic math to see if the antenna is fit for our operation.
- Aperture:
  The main objective here is how much signal we can gather or how big of a signal we can gather. Hence, it is advisable to choose antenna at the proper center frequency and whose front-facing diameter is good enough to accommodate all incoming signals in them at a time.
- Directivity and bandwidth:
  With the main beam pattern of the antenna, we need to see how much gain it has on its main lobe and how much suppression it provides on its side lobe. If the antenna has a good main lobe with 3dB cut off in its main lobe and complete suppression on its sidelobe, then it is perfectly fine to choose such antenna

- Effective length:
  With some special antennas like yagi and log periodic it is very much necessary to choose the proper length such that the antenna can operate with maximum efficiency in its center frequency and optimal efficiency in its other frequency.

Based on the requirement and type of assessment the type of antenna is selected. Below are some commonly used antenna for SDR

- Telescopic (Stock antenna with most of the SDR transceivers)
- Discone (Outdoor antenna)
- Vertical (Outdoor antenna)
- Yagi-Uda

5. Conclusion

We hope this gives you an introduction to the world of RF and SDRs. In this chapter, we covered all the basics and hardware tools required for an SDR assessment. In the next chapter, we will look at the software tools that are available and have a look at the classic approach to a target.

Chapter 11: Introduction to Software Defined Radio:Software tools and Approach Techniques

1. Introduction

In this chapter, we will be looking into some of the software SDR tools available out there. We'll also define an approach on how to go about an RF target.

2. Software

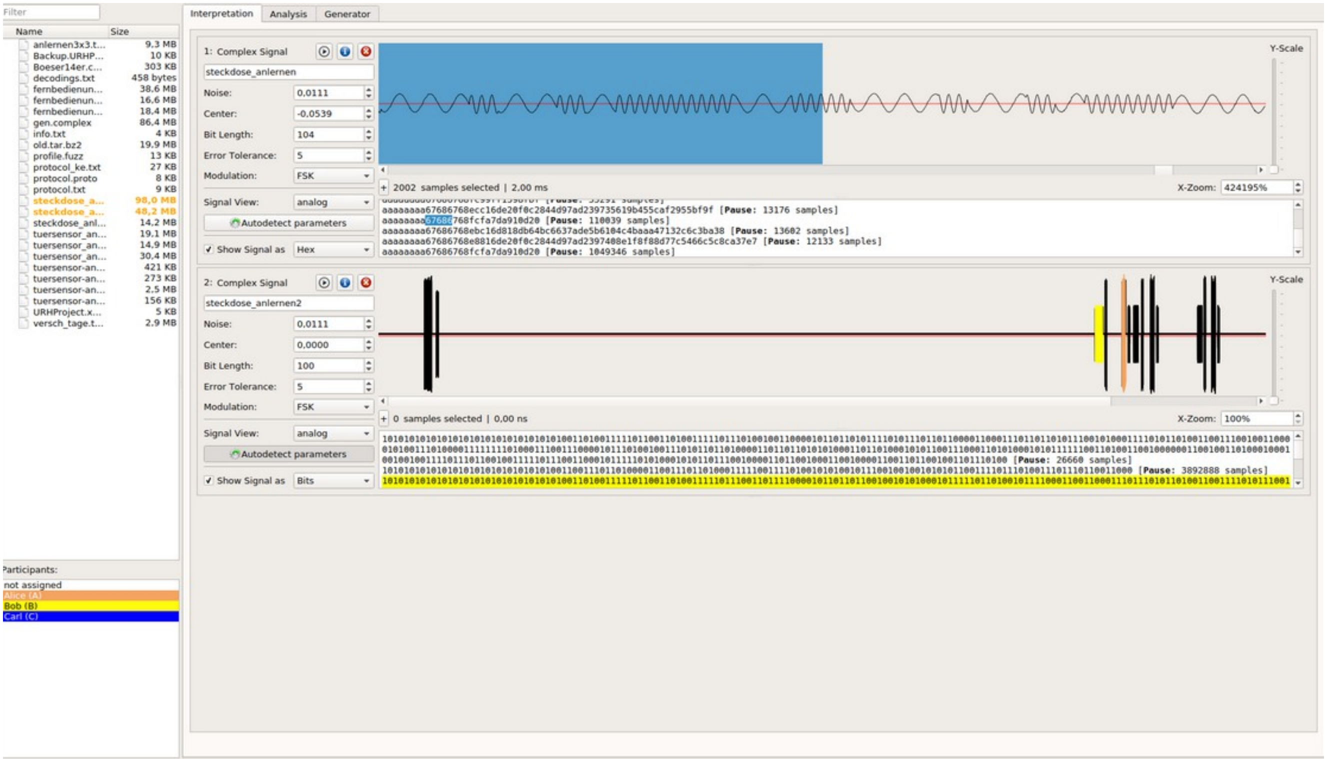
With a great open-source community, SDR has a variety of software tools with all signal processing functionality available. Let's look into some of the widely used SDR software available and what sets them apart. We'll be focusing on tools that are mainly available for Linux.

3. Recon tools

- GQRX:
  GQRX is a spectrum analyzer used for frequency band browsing and finding the operating frequency of the target. It comes with common demodulators like AM, CW, FM. Due to the demodulation functionality, it is possible to record demodulated signal streams which can be further analyzed in tools like Audacity and Inspectrum in the next phase of assessment. It is compatible with all major SDR hardware available. There are other alternatives to GQRX with more or less the same functionality, mentioned below:
- HDSDR/ SDR# (SDR-Sharp) [for windows]
- Qspectrum analyser (with automatic peak detection)
- Osmocom-FFT (spectrum analyzer included in the Osmocom GNU Radio blocks)

3.1 Basic assessment

- Universal Radio Hacker(URH):
  URH is a complete suite for wireless protocol investigation with native support to major SDR hardware. Almost everything is automated here, from spectrum analysis to even sending manipulated signals. One can effortlessly recognize the modulation type and get automatic decoding of the signal. For manual inspection, a differential view of received bitstreams is also there, which is very useful in interpreting the signal's data. Other major functionalities include the protocol analyzer (automated and manual). Here's where it gets interesting, It has a simulation environment for stateful attacks and a fuzzing element aimed at stateless protocols!



Name	Size
anlernen3x3.t...	9,3 MB
Backup.URHP...	10 KB
Boeser14er.c...	303 KB
decodings.txt	458 bytes
fernbedienun...	38,6 MB
fernbedienun...	16,6 MB
fernbedienun...	18,4 MB
gen.complex	86,4 MB
info.txt	4 KB
old.tar.bz2	19,9 MB
profile.fuzz	13 KB
protocol_ke.txt	27 KB
protocol.proto	8 KB
protocol.txt	9 KB
steckdose_a...	98,0 MB
steckdose_a...	48,2 MB
steckdose_anl...	14,2 MB
tuersensor_an...	19,1 MB
tuersensor_an...	14,9 MB
tuersensor_an...	30,4 MB
tuersensor-an...	421 KB
tuersensor-an...	273 KB
tuersensor-an...	2,5 MB
tuersensor-an...	156 KB
URHProject.x...	5 KB
versch_tage.t...	2,9 MB

1: Complex Signal
steckdose_anlernen
Noise: 0,0111
Center: -0,0539
Bit Length: 104
Error Tolerance: 5
Modulation: FSK
Signal View: analog
Autodetect parameters
☑ Show Signal as Hex

2002 samples selected | 2,00 ms

```
aaaaaaaa67686768ecc16de20f0c2844d97ad239735619b455caf2955bf9f [Pause: 13176 samples]
aaaaaaaa67686768fcfa7da910d20 [Pause: 110039 samples]
aaaaaaaa67686768ebc16d818db64bc6637ade5b6104c4baaa47132c6c3ba38 [Pause: 13602 samples]
aaaaaaaa67686768e8816de20f0c2844d97ad2397408e1f8f88d77c5466c5c8ca37e7 [Pause: 12133 samples]
aaaaaaaa67686768fcfa7da910d20 [Pause: 1049346 samples]
```

2: Complex Signal
steckdose_anlernen2
Noise: 0,0111
Center: 0,0000
Bit Length: 100
Error Tolerance: 5
Modulation: FSK
Signal View: analog
Autodetect parameters
☑ Show Signal as Bits

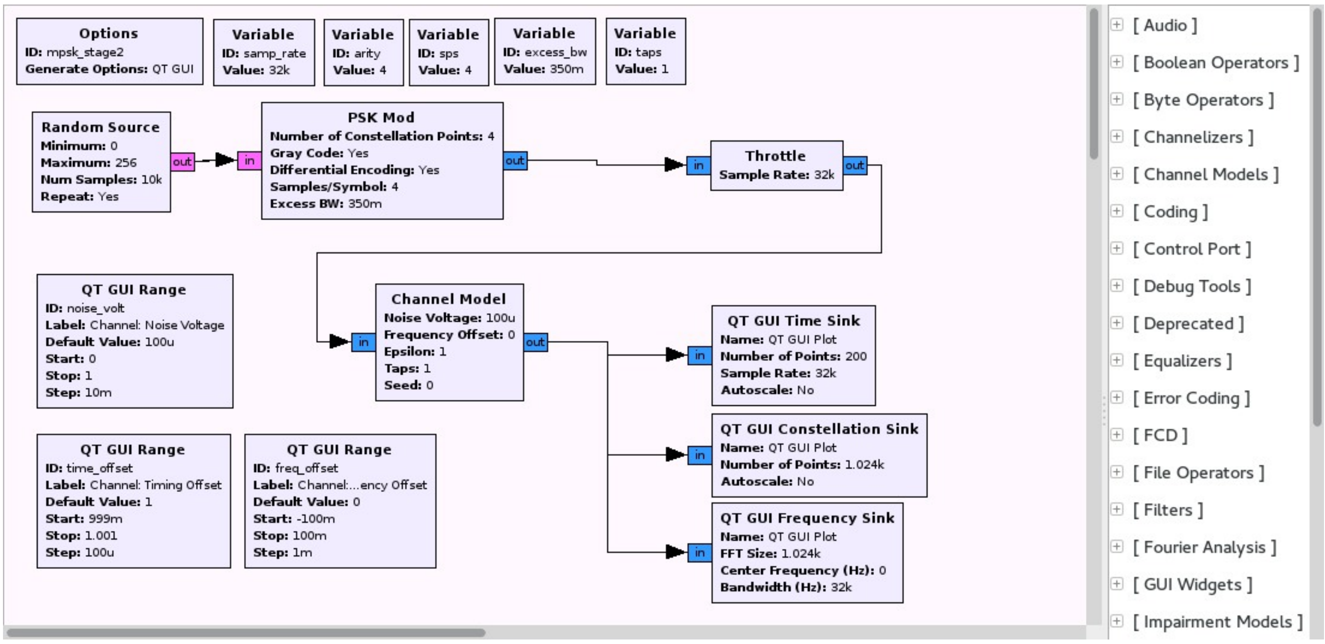
0 samples selected | 0,00 ns

Another alternative to URH is Inspectrum.

- Audacity:
  Audacity is a multichannel audio editing tool, but it turns into a radio signal analyzer when clubbed with GQRX. Audacity is open-source and is available for all common OS. It accepts only recorded signals; however the signal has to be demodulated, like a recorded signal from GQRX.

3.2 Advanced assessment

- GNU Radio
  GNU Radio is an open-source toolkit to implement SDRs. It provides basic blocks to perform different steps of signal processing, for example, filters, decoders, demodulators, and many more. It works with all of the major SDR hardware. The major benefit is the huge extensibilityof the framework. It is possible to write blocks in C++, or Python.
- GNU radio companion (GRC):
  GNU Radio Companion (GRC) is a frontend visualization tool that is part of the Gnu radi oframework. We should keep in the back of our mind that GRC was created to simplify the use of GNU Radio by allowing us to create python files graphically as opposed to creating them in code alone. It allows one to simply drag, modify parameters, and start processing the signal. We'll focus on it more as we proceed.



Options
ID: mpsk_stage2
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: arity
Value: 4

Variable
ID: sps
Value: 4

Variable
ID: excess_bw
Value: 350m

Variable
ID: taps
Value: 1

Random Source
Minimum: 0
Maximum: 256
Num Samples: 10k
Repeat: Yes

PSK Mod
Number of Constellation Points: 4
Gray Code: Yes
Differential Encoding: Yes
Samples/Symbol: 4
Excess BW: 350m

Throttle
Sample Rate: 32k

QT GUI Range
ID: noise_volt
Label: Channel: Noise Voltage
Default Value: 100u
Start: 0
Stop: 1
Step: 10m

Channel Model
Noise Voltage: 100u
Frequency Offset: 0
Epsilon: 1
Taps: 1
Seed: 0

QT GUI Time Sink
Name: QT GUI Plot
Number of Points: 200
Sample Rate: 32k
Autoscale: No

QT GUI Constellation Sink
Name: QT GUI Plot
Number of Points: 1.024k
Autoscale: No

QT GUI Range
ID: time_offset
Label: Channel: Timing Offset
Default Value: 1
Start: 999m
Stop: 1.001
Step: 100u

QT GUI Range
ID: freq_offset
Label: Channel:...ency Offset
Default Value: 0
Start: -100m
Stop: 100m
Step: 1m

QT GUI Frequency Sink
Name: QT GUI Plot
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 32k

3.3 Other Points of Interest

- Android:
  SDR is making its way into the mobile device as the processing capabilities of the mobile devices increases significantly over time. Although still very limited, but simply loading a few libraries of the device, connecting your SDR hardware via OTG cable to your android will do the job. Devices like RTL-SDR dongle, Lime SDR mini, and HackRF, and a few other work fine ewith the android devices. GNU radio companion (GRC): GNU Radio Companion (GRC) is a frontend visualization tool that is part of the Gnu radi oframework. We should keep in the back of our mind that GRC was created to simplify the use of GNU Radio by allowing us to create python files graphically as opposed to creating them in code alone. It allows one to simply drag, modify parameters, and start processing the signal. We'll focus on it more as we proceed.
- SDR touch:
  Similar to GQRX, it is used as a spectrum analyzer for the mobile device.
- GNU Radio Android:
  More Recently, GNU Radio for android came out. It's all your SDR solution in your mobile device. Although it has limited supported mobile devices as of now, major device coverage is expected over time.
- Scapy-radio:
  Scapy-radio is an extension to Scapy, an open-source network packet manipulation tool, written in Python. This extension uses Scapy as a back end for radio packet manipulation. As the gateway from Scapy to the SDR device, GNU Radio is used.

4 How to approach a target

We'll be breaking down how you can approach an RF target, capture, reverse-engineer it, and launch your attacks!

- SDR Hardware:
- HackRF One
- SDR Software:
- GQRX, GNURadio Companion
- Target:
  For this, we picked a locally manufactured $6 wireless doorbell, which turned out to be analog. Let's see how we go about it...

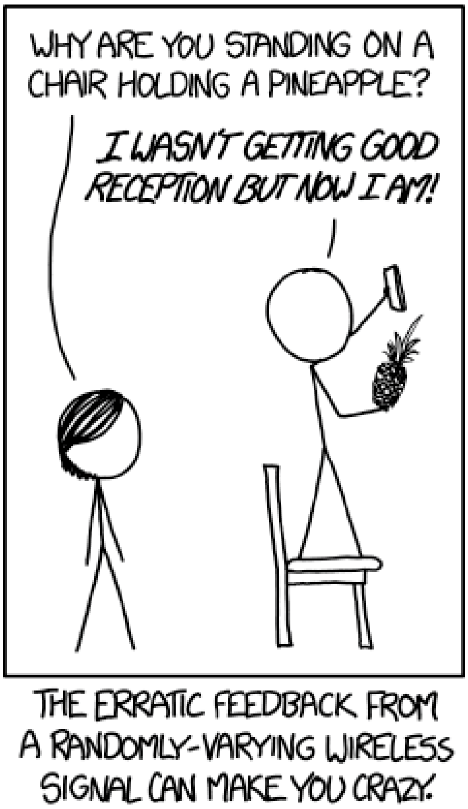

WHY ARE YOU STANDING ON A CHAIR HOLDING A PINEAPPLE?
I WASN'T GETTING GOOD RECEPTION BUT NOW I AM!
THE ERRATIC FEEDBACK FROM A RANDOMLY-VARYING WIRELESS SIGNAL CAN MAKE YOU CRAZY.

Image source:https://xkcd.com/1457/

4.1 Recon

In case you have the device, most of the task is done because you can simply look up the FCC ID of the device from here, which will give you a lot of details about the device i.e., operating frequency, the internals of the device, so on and so forth which will ease up a lot for the assessment.

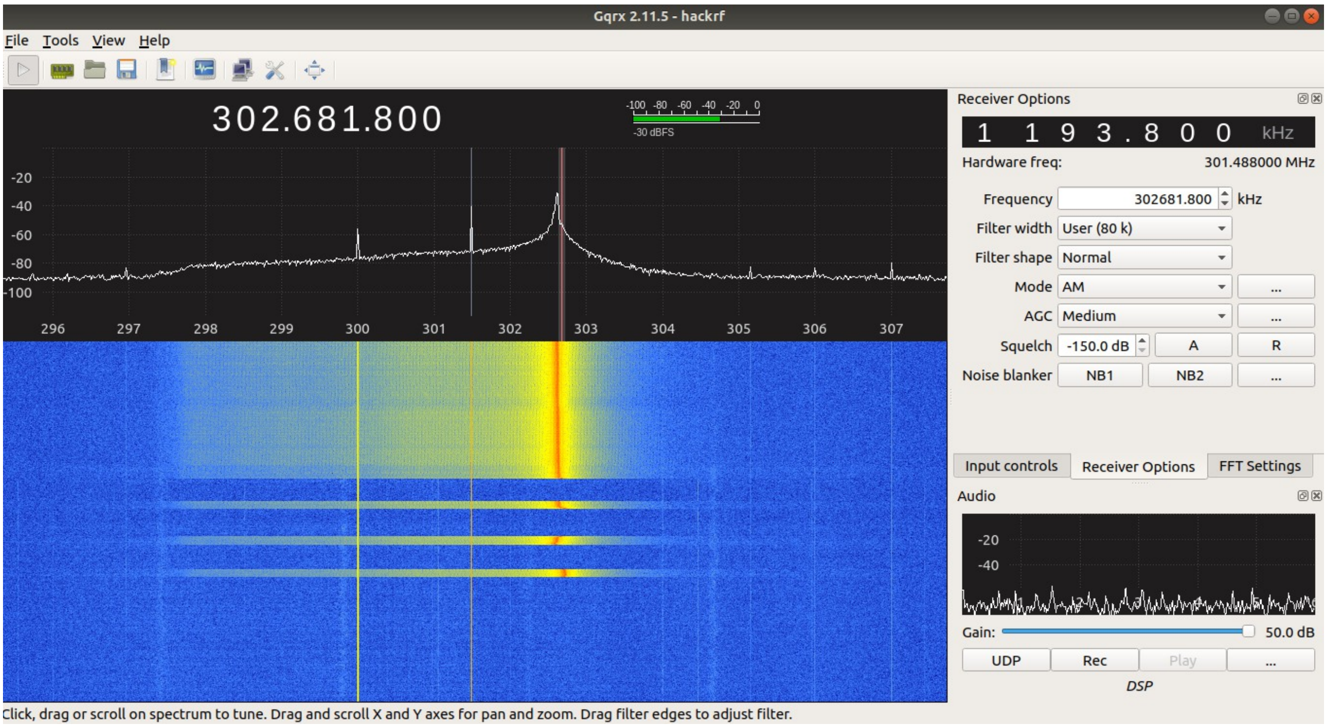
In our case, since it was a locally manufactured device, it didn't come with any FCC ID :(

4.2 Seek

You need to analyze the spectrum to find where the signal is being transmitted. You can use any spectrum analyzer you are comfortable with. Here's a list of common operating frequencies that will help you in finding the target frequency:

- 300MHz
- 433Mhz
- 868Mhz
- 915Mhz

All these are commonly used ISM and license-free bands. Bodies like WPC (Wireless planning and coordination wing of the Department of Telecommunications (DOT) of the Government of India and US Federal Communications Commission (FCC) allow such bands to be license-free provided that these will only be used by low powered wireless equipment with specifics defined for bandwidth, output power and maximum effective radiated power.



Gqrx 2.11.5 - hackrf

Hardware freq: 301.488000 MHz

Frequency: 302681.800 kHz

Filter width: User (80 k)

Filter shape: Normal

Mode: AM

AGC: Medium

Squelch: -150.0 dB

Gain: 50.0 dB

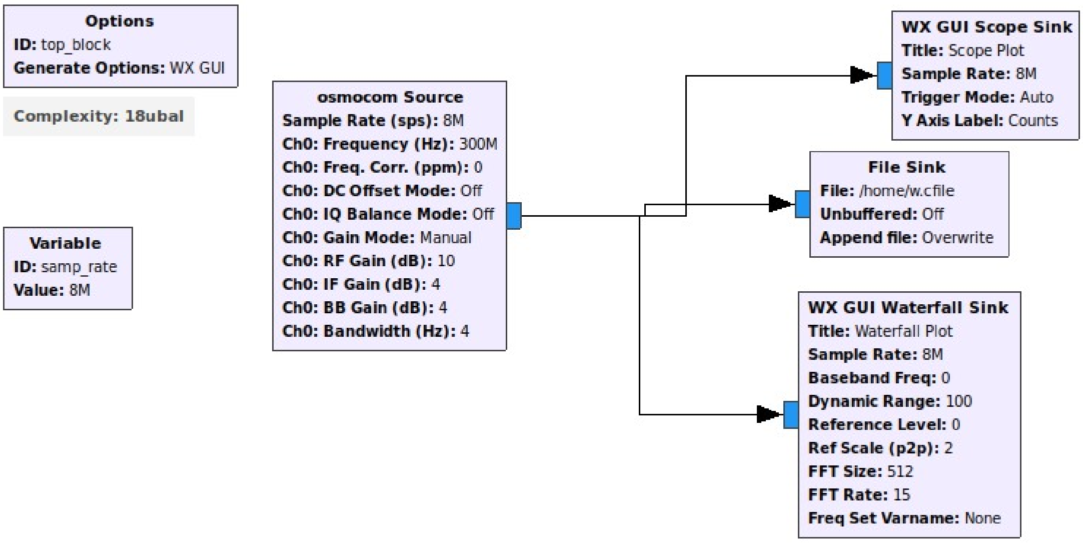
Click, drag or scroll on spectrum to tune. Drag and scroll X and Y axes for pan and zoom. Drag filter edges to adjust filter.

File
Tools
View
Help
302.681.800
-100 -80 -60 -40 -20 0
-30 dBFS
-20
-40
-60
-80
-100
296
297
298
299
300
301
302
303
304
305
306
307
Receiver Options
1 1 9 3 . 8 0 0 kHz
Noise blanker
NB1
NB2
A
R
...
Input controls
Receiver Options
FFT Settings
Audio
-20
-40
UDP
Rec
Play
DSP

In our case, we used GQRX for this step. We observed the peak at 302MHz, which is a littl unusual for any operating device.

4.3 Record

Capturing the signal is a good practice to preserve the signal for analysis. One can easily run signal processing operation on a recorded signal, without the need for any RF target producing a live signal. In our case, we are using the below GRC flowgraph. We saved the signal in a .cfile.



Options
ID: top_block
Generate Options: WX GUI

Complexity: 18ubal

Variable
ID: samp_rate
Value: 8M

osmocom Source
Sample Rate (sps): 8M
Ch0: Frequency (Hz): 300M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 4
Ch0: BB Gain (dB): 4
Ch0: Bandwidth (Hz): 4

WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 8M
Trigger Mode: Auto
Y Axis Label: Counts

File Sink
File: /home/w.cfile
Unbuffered: Off
Append file: Overwrite

WX GUI Waterfall Sink
Title: Waterfall Plot
Sample Rate: 8M
Baseband Freq: 0
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Freq Set Varname: None

Recording the analog doorbell signal (flowgraph)

4.4 De-Modulation

The best way to do it is just by simply looking at the signal. If you're well versed in the modulation techniques you can easily understand the type of modulation being used i.e. compressions and decompressions in FSK. Below are some commonly used modulations:

Chapter 11: Introduction to Software Defined Radio: Software Tools and Approach Techniques

- FSK (Frequency shift keying)
- ASK (Amplitude shift keying)
- OOK (On-off keying)
- PSK (Phase shift keying)

4.5 Process

Once we have figured out the modulation and operating frequency, we can start to process the signal using the GRC blocks and creating flow graphs. This usually includes things like demodulating the live/captured signal, amplifying the signal, porting it to Wireshark, and so on. We will discuss GRC in much more detail as we proceed covering things like usage of GRC blocks required to process a specific signal.

4.6 Decode

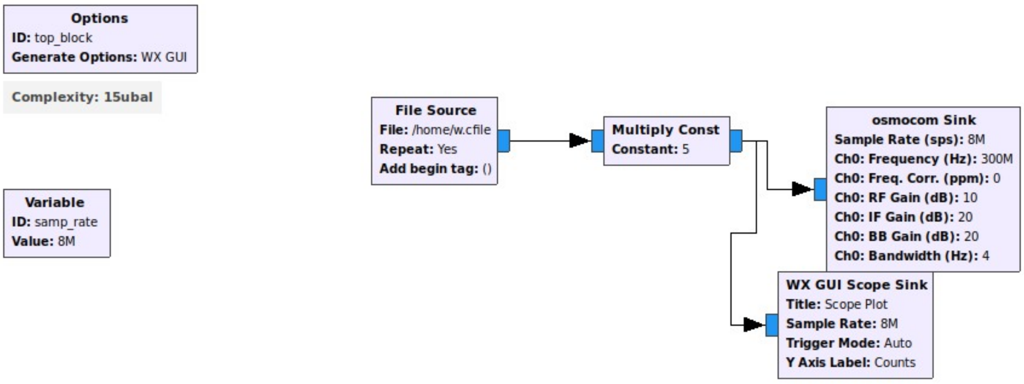
So once you get your hands on the bits/bitstream, you can start decoding it. Commonly used encoding techniques are

- NRZ
- NRZI
- Bipolar AMI
- Manchester

To name a few. You'll find them already present and ready to use in Universal Radio Hacker(URH).

4.7 Attack!

Once you are done reversing your signal i.e. figuring out things like modulation, encoding, data bytes, and other specifics. You can launch your attacks now! The most common one is the replay attack, wherein one sends back the captured signal.
Steps 3 & 5 are not applicable for our target (since it is analog), but we have an attack for our target, we'll do a replay attack using GRC. Below is the respective GRC flowgraph. We replayed the captured signal .cfile.



Options
ID: top_block
Generate Options: WX GUI

Complexity: 15ubal

Variable
ID: samp_rate
Value: 8M

File Source
File: /home/w.cfile
Repeat: Yes
Add begin tag: ()

Multiply Const
Constant: 5

osmocom Sink
Sample Rate (sps): 8M
Ch0: Frequency (Hz): 300M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20
Ch0: Bandwidth (Hz): 4

WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 8M
Trigger Mode: Auto
Y Axis Label: Counts

Replaying the analog door bell signal (flowgraph)

5. Conclusion

We hope you got a clear understanding of what are the major SDR software tools available and why is one better than the other. Also, a sneak-peak into how to approach an RF target would have given you an idea of the steps involved in an RF target assessment using SDR.

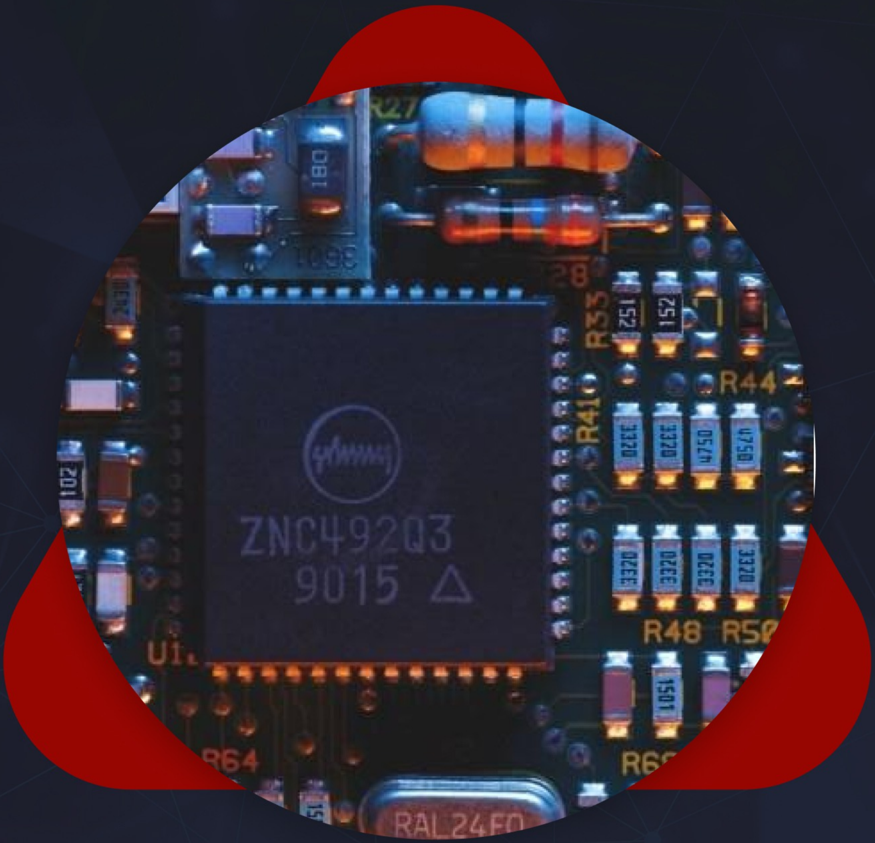
Reference :

Chapter 10 : Introduction to Software Defined Radio: Hardware tools

- Getting started with Radio Hacking – Part 1 – Radio Frequency basics and theory
- Getting Started with Radio Hacking – Part 2 – Listening to FM using RTL-SDR and GQRX
- SDR with HackRF by Michael ossmann
- University of Illinois Digital signal processing series

Chapter 11 : Introduction to Software Defined Radio: Software tools and Approach techniques

- GNURadio-Android



Part 4

FIRMWARE

Chapters: 12

What to expect:

Get a run-through of the different components involved in building IoT Firmware including interfaces, operating systems, and reliable open-source tools used for reverse engineering.

Chapter 12 : Firmware Reverse Engineering

Firmware is the software counterpart of IoT devices. There are many technologies involved in building a functional IoT Firmware, and there are many vendors contributing to the development of these technologies, some of which are Cisco, Linux, Wind River, etc. In this chapter, when I say firmware, I will mostly be referring to the software component of the device. Firmwares have varying complexity from a Bare-Metal firmware driving tiny, less powerful micro-controller to microprocessor-based full-fledged Operating systems like Linux, which is used in more complex devices like routers, television, etc.

In this chapter, we will look at different components involved in building firmware, and then we will discuss how you can use various open-source tools to reverse engineer firmware. Reversing is an important step that will help you to do further analysis of the device and the firmware. So let's start looking at what bare-metal firmware are:

1. Bare Metal Firmware

Depending upon the device's application, the technology stack is used. So, if you consider a device whose task is to read and report surrounding temperature and humidity, it won't need something very complicated software, like Linux. For those cases, Bare-Metal firmware is used. Now, what is Bare Metal Firmware, you might Ask? In simple terms, this type of firmware directly interfaces with the hardware, there is no driver or kernel involved.

Bare Metal firmware doesn't do much complicated things, they are usually tasked with not more than 3 or 4 tasks, and those tasks are put in the loop as they are scheduled to run in specific order/condition. The SDK's provided by the vendor for these devices provide life cycle methods that programmer write those functions are run in the loop. Some little complex feature variants of these SDK's are FreeRTOS, mbed-os which are real-time operating systems that allow you to do task scheduling and have a very quick response to some of the interrupt requests.

Base Metal firmwares are written in C, so all attacks like buffer overflow related vulnerabilities are also valid for these firmwares. These devices usually collect data and send it to the central server or communicate with other devices connected via UART/SPI Bus(its peripherals). There are chances of finding the vulnerability in the communication protocol, incorrect handling of packets, key exchange, buffer overrun, etc.

Micro-controller based devices are not just used in sensor networks; they are everywhere, from the refrigerator, microwave oven, security alarm system. Your car has dozens of these, and so does your laptop/computers. Usually, when you don't have source code for this firmware, you take the approach of reverse engineering.

Reversing these binaries is a little different from reversing Windows EXE and Linux ELF files; they then don't have a predefined structure. For bare-metal binaries, you need the data-sheet for the chipset and create a memory-map in your disassembly tool like IDA, Ghidra, etc., to get a proper disassembly. Another very critical question memory-map also helps users to answer is what GPIOs, other peripherals is the device interacting with? This information will help to understand the functionality of the device. A lot more can be said on this, but I want to keep this chapter crisp, so for more details, visit this link. Now let's look at what is a fully-blown operating system based firmware.

2. Fully Fledged Operating System

When you see more complex systems like Routers, Smart Home Dashboard, Drones, and healthcare devices, they do little more than what bare-metal systems do. Take, for example, a typical wireless router that has features like connecting by LAN and WIFI; they can blacklist/whitelist specified MAC addresses, and some modern routers have antivirus software and other protection features. To implement all these functions, you need a full-fledged Operation System to support all these sophisticated features.

There are many different Operating System options for embedded systems, like Linux, Windows CE, Cisco IOS, Symbian, Android, VxWorks, etc. Some of them are special-purpose like Cisco for routers, and most of them are more general purpose. General statics say that the most popular choice of OS amongst embedded system products is Linux. There are many reasons for that, some of which is its open-source nature, flexibility, and most importantly, it's free. You can find Linux powered Devices around like internet routers, infotainment systems, etc.

This kind of firmware is usually packaged with at least three components, Bootloader, Kernel, and the file-system. The Bootloader is the piece of software that helps in loading Operating System, Kernel and passes various information needs. Once the bootloader is done executing, the Kernel takes over. Once the kernel has started executing, it starts other user applications to make an Operating System usable by the end-user. This usually involves starting various applications and services in the background. Once all this is done then, the user can interact with the system. All the user application and app data are stored on the file-system.

Security assessments of these devices usually start with auditing applications running on these devices; Most juicy targets are the remote application services. Issues discovered in these services have a high impact on the security of the device as attackers don't need to be anywhere near it to compromise it. Anyone on the internet can take control of it if they can exploit the vulnerability. These services are usually web-servers used to configure and control the device features or other services like UPnP, which help in discovering the device by other devices.

3. Security Tools

Now that we have a high-level overview of how firmware looks like, the next question that comes to your mind is how do you start dissecting the firmware and start analyzing the firmware. An analysis can broadly be classified into the static and dynamic analysis. In static analysis, you read the code and look for bugs. If you don't have the source code you start reverse-engineering the binary and read the assembly instructions to understand the functionality. While on the other side, dynamic analysis involves running the application and observing its behaviours. When analyzing a firmware, I usually used a combination of both the analysis techniques to achieve my goal. Below is the list of some of the tools which I frequently use:

3.1 Binwalk

This perhaps is one of the most popular tools for unpacking firmware. It is a firmware extraction tool, it tries to crave out binaries inside any binary blob. It does this by searching for signatures for many of the common binary file formats like zip, tar, exe, ELF, etc. Binwalk has the database of binary header signatures against which the signature match is done. The common objective of using this tool is to extract the file system like Squashfs, yaffs2, Cramfs, ext*fs, jffs2, etc., which is embedded in the firmware binary. The file system has all the application code that will be running on the device. This tool also has many parameters that you can tweak to make extraction better. You can visit this link to read more details about what are different parameters and how to use them.

3.2 Qemu

This is a valuable tool for people working on cross-architectures (like ARM, MIPS, etc.) environment, which usually is the case for embedded developers. This tool provides a way to emulate binary firmware for different architectures like ARM, MIPS, etc. on the host system, which is of different architectures like x86, amd64. This kind of tool is handy when you want to audit a firmware, but you don't have the device or setting up a debugger for that system is very difficult. Qemu can help you to do full-system emulation, or a single binary emulation of ELF file for the Linux system and many different platforms. You can check this link to get more details about the project.

3.3 Gdb-multiarch

GDB is a dynamic debugging tool used when you want to pause a running process and inspect its memory and registers. However, gdb supports just the architecture for which it is compiled. But when you want to debug the application of different architecture for cross-architecture support, you will need its sister project gdb-multiarch, which helps you to do cross-architecture debugging.

3.4 Firmware ModKit

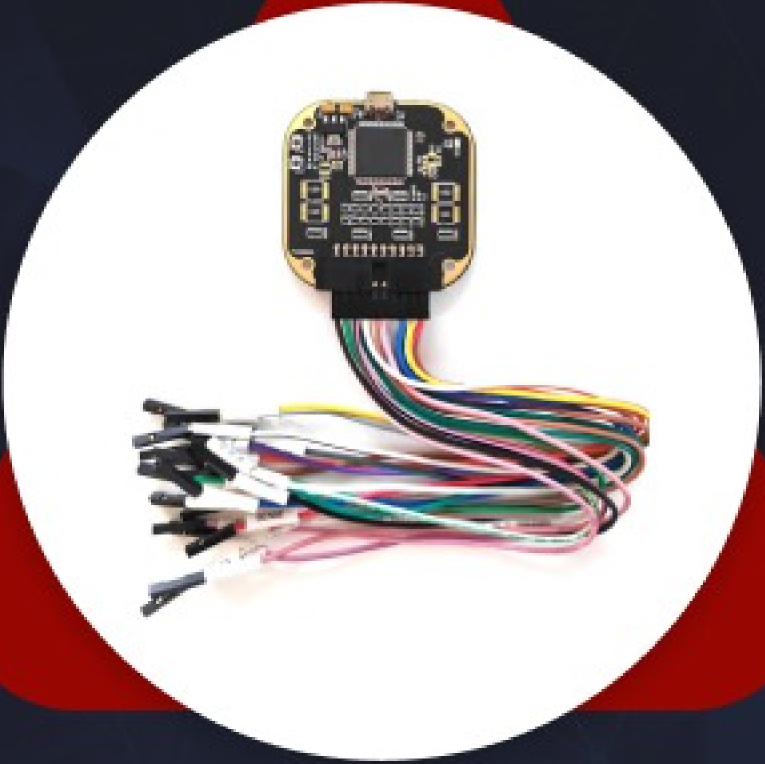
This tool helps you to patch the firmware and repackage it. It extracts the firmware using Binwalk and gives you a directory that has the firmware file-system laid out. You can then patch whatever you want, add/delete a file or patch an existing and the ModKit can pack it back up such that you can flash the new firmware on the device storage and boot up the newly patched firmware. You can download the code for this tool from this link.

My workflow when working with firmware involves all of these tools. The first step usually starts with unpacking the firmware with Binwalk. Next, try to emulate the application of interest in the firmware with the Qemu. If I am not able to emulate the binary, then I investigate the issue with gdb and patch it so that Qemu can run it and look for security issues. This setup also helps me to fuzz the service.

4. Conclusion

We looked at what the firmware is and what are different types of IoT Firmware you will encounter when assessing an IoT Device. We gave you an overview of how to deal with a different kind of firmware, and we also looked at how different tools can help us to security analysis of a firmware.

Reference :



Part 5

HARDWARE

Chapters: 13 to 20

What to expect:

Equip yourself with the basics of Hardware Recon and Debug ports as we also dive into learning about different Hardware attack surfaces like SPI, I2C, UART, JTAG, SWD, SCAA, and FI.

Chapter 13: Introduction to Hardware Recon

In this chapter, we will discuss how to perform reconnaissance on hardware. The hardware comprises physical electronics components that include micro-controller, microprocessor, resistor, capacitor, LEDs, etc., which make a device work. The hardware reconnaissance process consists of several steps like device disassembly, looking at the various components, debug ports like UART, JTAG/SWD, identifying the chips, finding the respective datasheet, and finding the information from the datasheet. So let's get into each of them.

1. Importance of Hardware Recon

During the process of pentesting or security research in general, before starting to attack the target we have to have a complete understanding of the system even if it is a blackbox. The recon process helps in identifying different components that make up the system so we can gear our attacks towards what we know and what loopholes we have identified while understanding the system. During the recon, we identify pieces and connect those pieces to derive intelligent information that will help in compromising the security of the system. When attacking a hardware device, we need to first understand what is the hardware made up of i.e. what are the different components of the hardware to be able to create an effective threat model and mis-use cases to be used while finding vulnerabilities. We will see below what is the basic information derived during this process.

Bare Metal firmware doesn't do much complicated things, they are usually tasked with not more than 3 or 4 tasks, and those tasks are put in the loop as they are scheduled to run in specific order/condition. The SDK's provided by the vendor for these devices provide life cycle methods that programmer write those functions are run in the loop. Some little complex feature variants of these SDK's are FreeRTOS, mbed-os which are real-time operating systems that allow you to do task scheduling and have a very quick response to some of the interrupt requests.

Base Metal firmwares are written in C, so all attacks like buffer overflow related vulnerabilities are also valid for these firmwares. These devices usually collect data and send it to the central server or communicate with other devices connected via UART/SPI Bus(its peripherals). There are chances of finding the vulnerability in the communication protocol, incorrect handling of packets, key exchange, buffer overrun, etc.

2. Hardware Tools

We will need some key pieces of physical equipment to perform hardware reconnaissance.

1. Multimeter:
   A multimeter is a very important tool for circuit probing. It will help us to test all the components and to measure resistance, voltage, and current level and electric continuity between two points.
2. A soldering iron, Solder, Flux, Tweezer, Soldering wick, Cutter, Wire stripper:
   These are soldering tools, useful to add and remove the components from the PCB.
3. Screwdriver set:
   Necessary for disassembling the device. Nowadays device disassembly is quite a tough job. Sometimes, manufacturers use tamper protection to prevent people from gaining access to internal components of the device.
4. Jumper wires:
   Useful to connect two devices electrically.
5. Desoldering Pump/Hot Air Rework:
   The Desoldering pump requires removing SMD components without destroying the PCB at a suitable temperature.
6. Magnifying Glass:
   Useful to see the components clearly and helps in recognizing the components model, make, and part numbers. Usually, they are written in very small sizes that are difficult to read with the naked eye.
7. Vise Stand:
   Useful to hold PCB while soldering or desoldering components. or while inspecting PCB

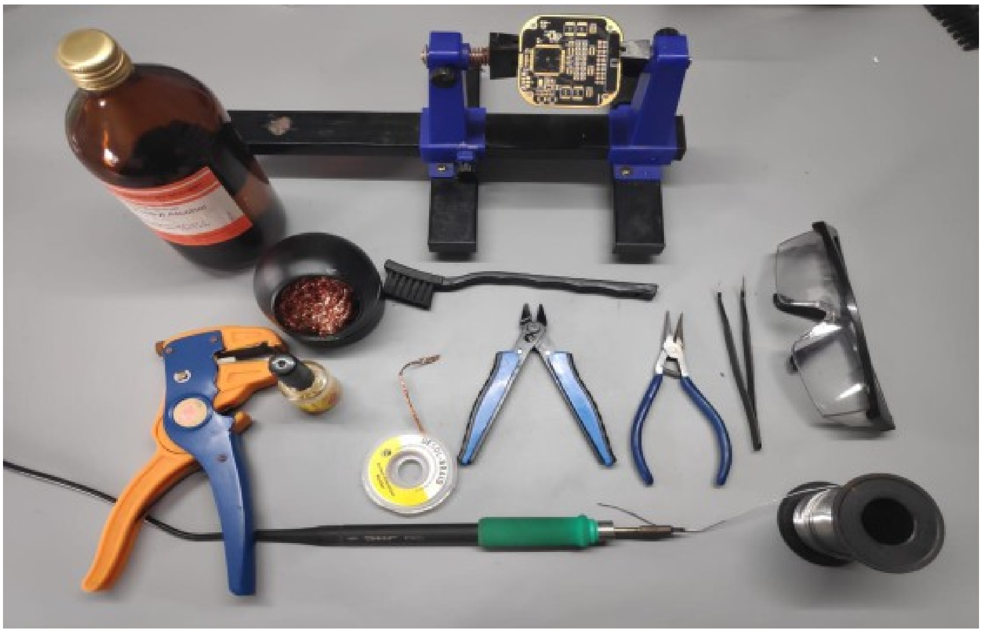


Fig 1. Hardware tools

3. Basic knowledge of electronics

Basic Electronics is one of the most important things to understand if you want to get into hardware hacking. You will need to understand what is happening in the device and how any given component can be exploited. So here we will look at some basic electronic components and the purpose of using them in circuits.

1. Resistor: It adds resistance between two components. It is measured in ohms.
2. Capacitor: It charges and discharges in specific interval of time and used to stabilize the power supply in Circuit. It is measured in farad.
3. Inductors: They are used for filtering and smoothing high-frequency noise in the circuit using electromagnetic discharge. It is measured in Henry.
4. IC: Integrated Circuits is a set of electronic circuits on small pieces of silicon.
5. LED: Light Emitting Diode.
6. EEPROM (Electrically Erasable Programmable Read-Only Memory): Embedded devices use these as a means of storage.
7. Crystals: These oscillate at a given frequency, similar to a timer.
8. Transformers: They are used to convert voltage levels. Mostly used for converting AC mains to DC supply with some extra circuitry.
9. Diodes: Used to restrict current flow in one direction.
10. Relay: It is a switch that controls (open and close) circuits electromechanically.
11. Microcontroller/Microprocessor: It is a tiny little computer on a single metal-oxidesemiconductor (MOS) integrated circuit (IC) chip.
12. SoC (System on Chip): They can be just a Processor or Processor + memory + peripherals.
13. Transistor: It is used to amplify and switch the signals and electrical power.
14. Battery: It converts chemical energy into electrical energy.
15. Motor: It converts electrical energy into mechanical energy.
16. Switch: It interrupts the current.
17. PCB: Printed circuit board (PCB) is a non-conductive material with conductive lines printed or etched.

4. Different types of packages of components

The packages are divided into two types based on how they are mounted on a Printed circuit board(PCB).

4.1 Through-hole mount package

This type of component is designed in such a way that the pins of the component are inserted in the PCB, and another side of the PCB pin can be solderable. These types of components are bigger than surface-mount packages. These components are cheaper as compared to SMD components. See the different types of Through-hole gull-wing components below the photo.

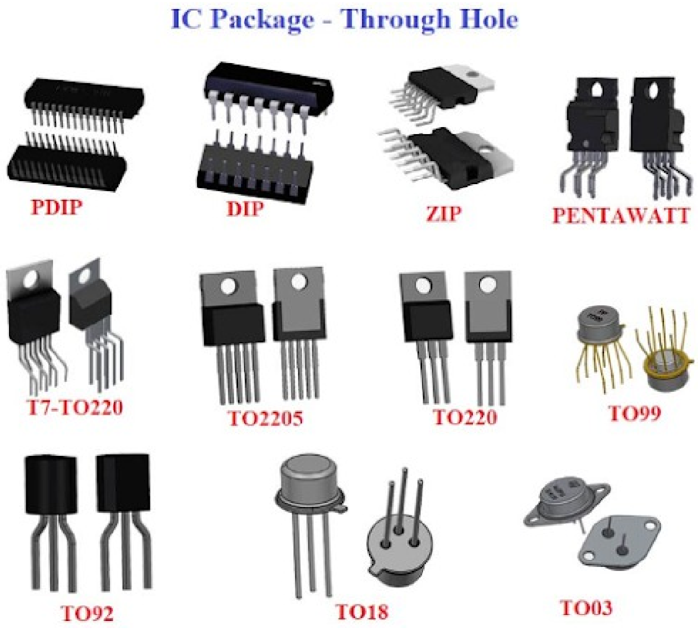

IC Package - Through Hole
PDIP
DIP
ZIP
PENTAWATT
T7-TO220
TO2205
TO220
TO99
TO92
TO18
TO03

Fig 2. IC Package-Through-hole
Image Source

4.2 Surface Mount Package

The surface-mount package component mounts directly on the PCB surface. This type of component takes minimum size and takes less time to assemble, but it also increases the chances of defects. Below are the type of SMD package components

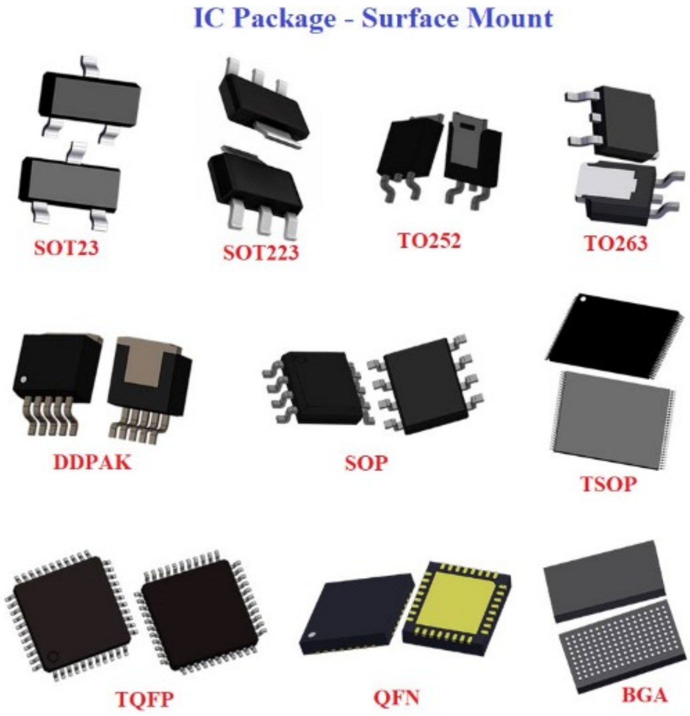

IC Package - Surface Mount
SOT23
SOT223
TO252
TO263
DDPAK
SOP
TSOP
TQFP
QFN
BGA

Fig 3. IC Package-Surface Mount
Image Source

4.3 Types of SMD components

- Small Outline L-leaded Packages (SOP)
  This type has gull-wing type leads that come out from the body in the L shape and can be mounted directly on the PCB
- Quad flat L-shape packages (QFP)
  This is the same to SOP only difference is that pins come out from 4 directions instead of 2 and can be mounted directly on the PCB.
- Ball Grid Array(BGA)
  These types of packages have a solder ball array on the backside of the component.

5. Printed Circuit Board (PCB)

Now we have a good understanding of electronic components and their different packages. So what next?

Here we will discuss Printed circuit board (PCB) is a nonconductive material with conductive lines printed or etched. Electronic components mounted on the PCB board such as Transistors, resistors, capacitors, integrated circuits(ICs), and the traces connect the components to form a working circuit. PCBs are most commonly made of fiberglass, composite epoxy, or other composite material. PCBs are made up of multiple layers. Even a simple single-sided (one layer) board is made up of a conductive metal layer and a substrate layer composited together. As the complexity of the PCB increases, so will the number of layers within it.

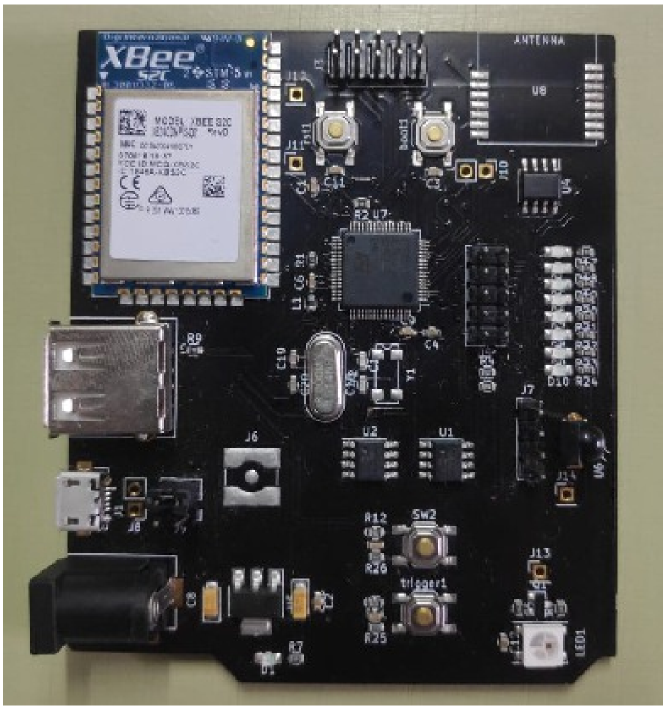


Fig 4. PCB Top side

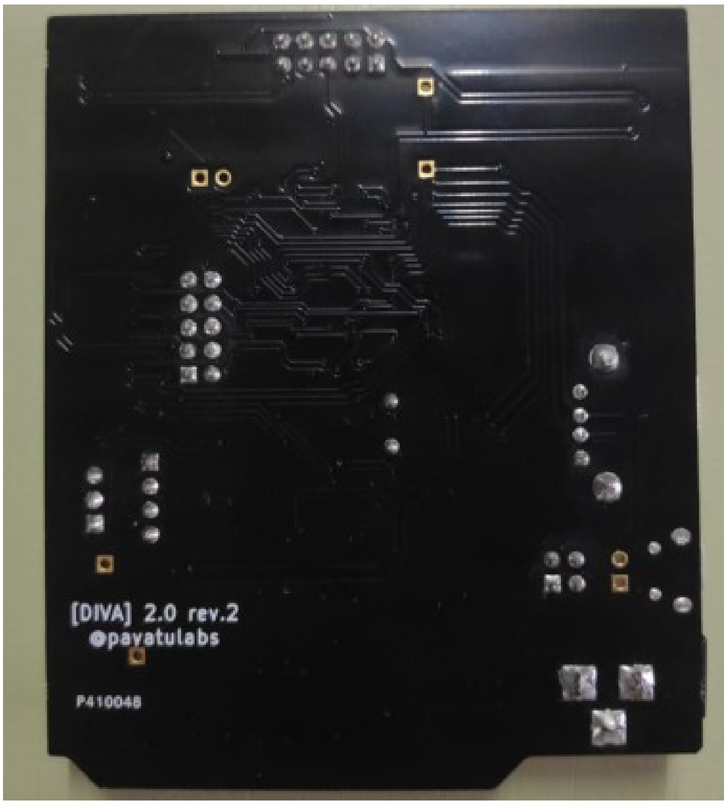


Fig 5. PCB Bottom side

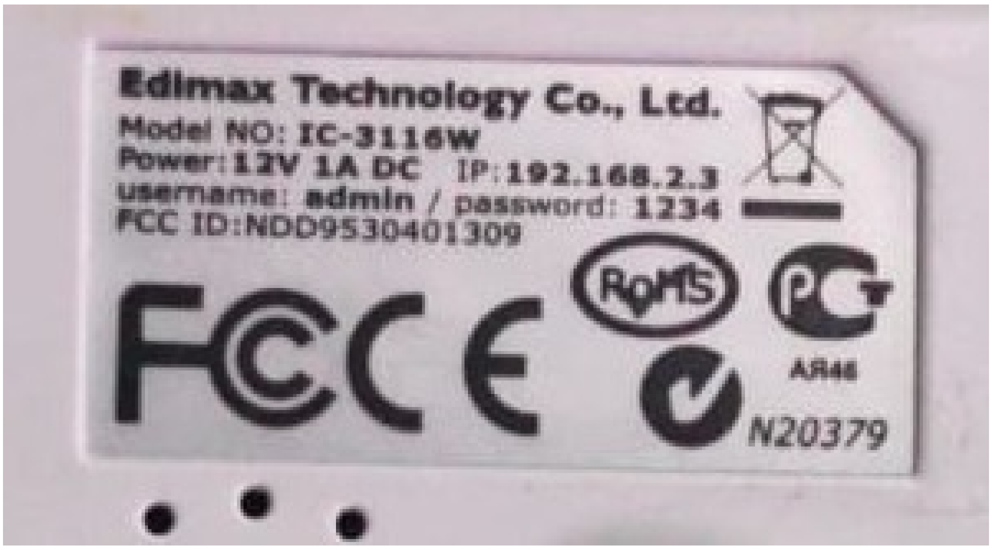
There are a lot of software tools available for designing a PCB like Kicad, Eagle, Orcad. We use Kicad for PCB design. It's easy to use and has the feature to view PCB in 3D. PCB gives us a lot of information like different components, debug ports, test pads, etc. used. It helps researchers create a threat model/mind map of attack scenarios for the device.

6. FCC ID

An FCC ID is a unique identifier assigned to hardware products registered with the United States Federal Communications Commission (Hence, the name FCC ID).

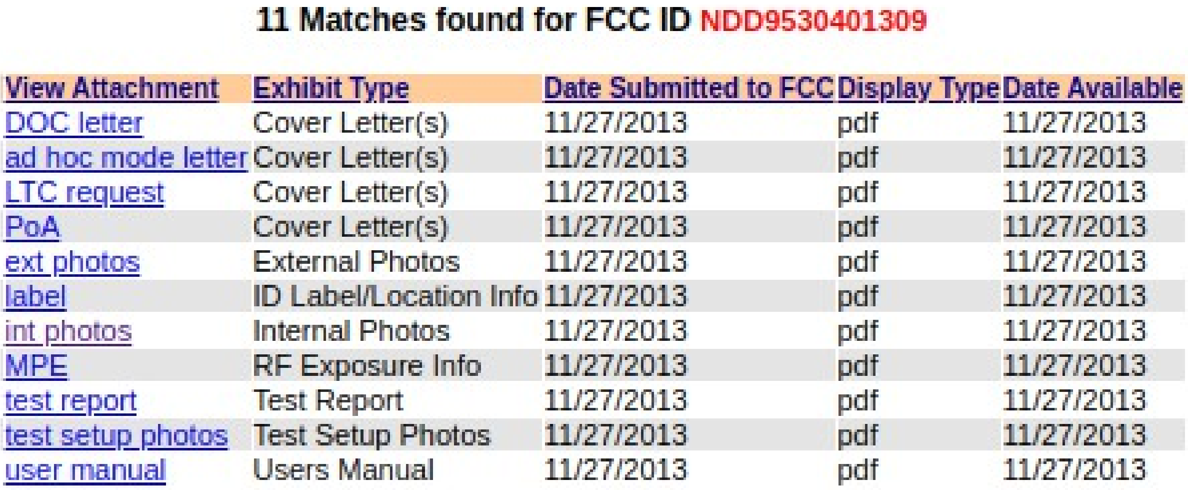
Most of the smart devices have an FCC ID which will give you a lot of details about the device like internal photos of the device, operating frequency, etc. This information is crucial for identifying the components as well as understanding the RF technology used in the device. You can search for the details of the devices by their FCC IDs on fcc.gov.

Let's search for the details of an Edimax Camera device. For example, its FCC ID is NDD9530401309. As you can see, we get a lot of interesting device-related information.



Edimax Technology Co., Ltd.
Model NO: IC-3116W
Power: 12V 1A DC IP: 192.168.2.3
username: admin / password: 1234
FCC ID: NDD9530401309

Fig 6. Edimax Camera



11 Matches found for FCC ID NDD9530401309

View Attachment	Exhibit Type	Date Submitted to FCC	Display Type	Date Available
DOC letter	Cover Letter(s)	11/27/2013	pdf	11/27/2013
ad hoc mode letter	Cover Letter(s)	11/27/2013	pdf	11/27/2013
LTC request	Cover Letter(s)	11/27/2013	pdf	11/27/2013
PoA	Cover Letter(s)	11/27/2013	pdf	11/27/2013
ext photos	External Photos	11/27/2013	pdf	11/27/2013
label	ID Label/Location Info	11/27/2013	pdf	11/27/2013
int photos	Internal Photos	11/27/2013	pdf	11/27/2013
MPE	RF Exposure Info	11/27/2013	pdf	11/27/2013
test report	Test Report	11/27/2013	pdf	11/27/2013
test setup photos	Test Setup Photos	11/27/2013	pdf	11/27/2013
user manual	Users Manual	11/27/2013	pdf	11/27/2013

Fig 7. FCC ID Table

7. Disassembly

Depending on the device that has been put together, we have to use appropriate tools to take apart all different parts. We also recommend that you have a good screwdriver toolkit for the entire hardware assessment process, as varying devices will have different kinds of screws used in them. So, here we will perform hardware recon on the ASUS RT-N10E Wi-Fi router. We will disassemble the router to access the Printed circuit board. There are 4 screws on the backside of the device that should be removed, then the plastic enclosure can pull apart.

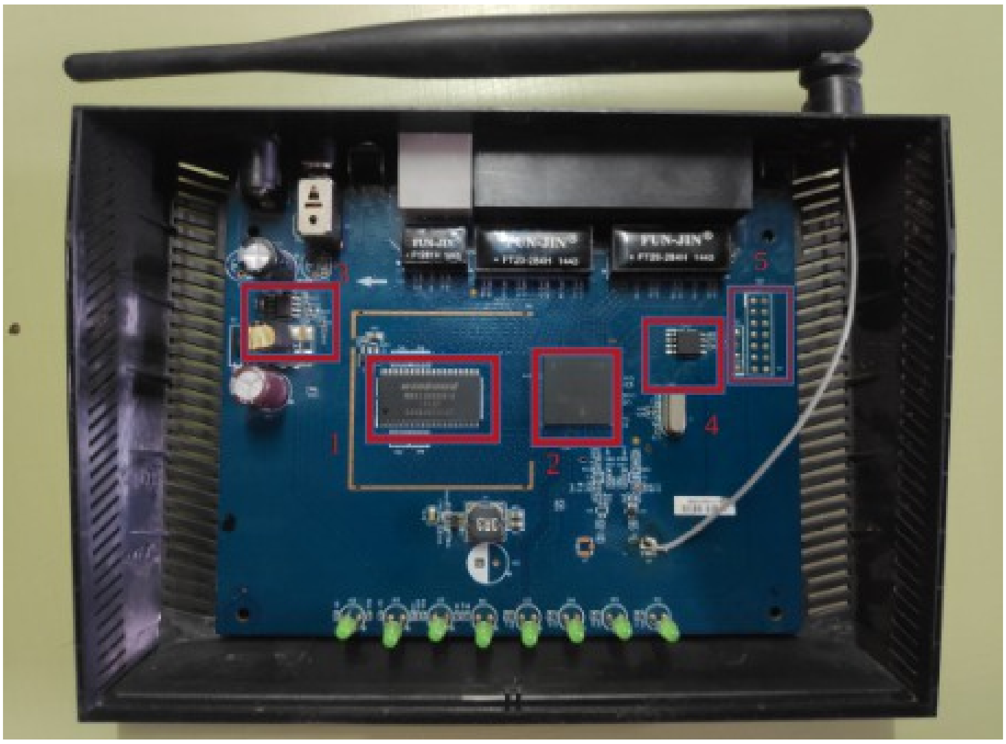


Fig 8. ASUS router

7.1 PCB Board Analysis

This is a very simple board with very few components and a header for debugging. More complicated devices will have many different components, multiple processors, CLPDs and/or FPGAs, multiple flash memories for different firmware and settings,cryptography co-processors, and various debugging ports and interfaces. In this PCB board, there are 4 ICs that we highlighted in the red box and some pinouts(headers). We need to identify what the ICs are and what are the pinouts, and why they are important:

1. WINBOND
2. BROADCOM
3. Fiti power
4. Macronix
5. Pinouts (Header)

7.2 Chip Analysis

Let's take a closer look at the ICs and see what we can unearth.

7.2.1 WINBOND

- Here we can see clearly what is written on the IC.
- Winbond is the manufacturer, and W9812G6KH-6 IS the Part number of the IC.

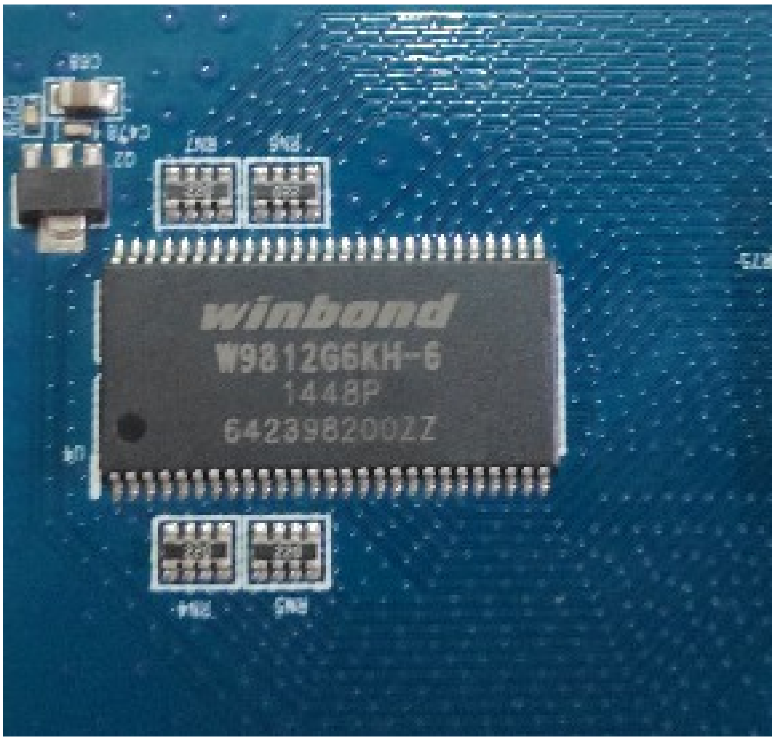
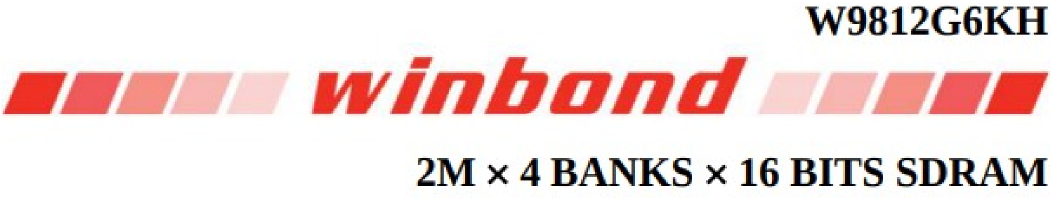


Fig 9. Windbond IC

- Let's Google the part number and download the datasheet. [datasheet]



W9812G6KH

winbond

2M × 4 BANKS × 16 BITS SDRAM

Fig 10. Datasheet

From the datasheet, we get to know that W9812G6KH is a high speed synchronous dynamic random access memory. We will get more information from the datasheet like memory size, clock frequency pin diagram power supply, packages.

7.2.2 BROADCOM

- The part number of the IC is BCM5356 KFBG.

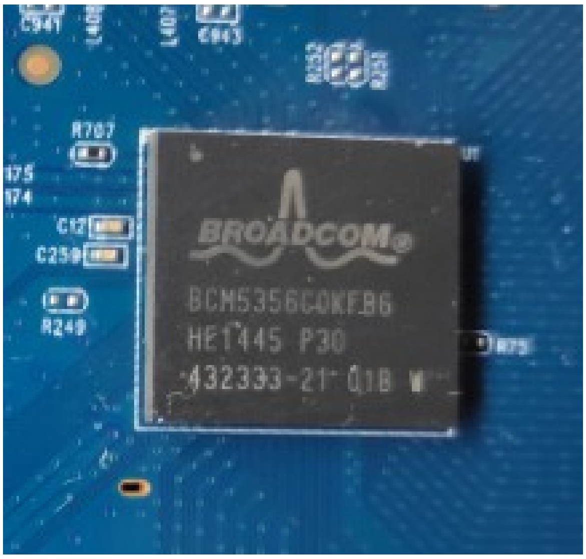
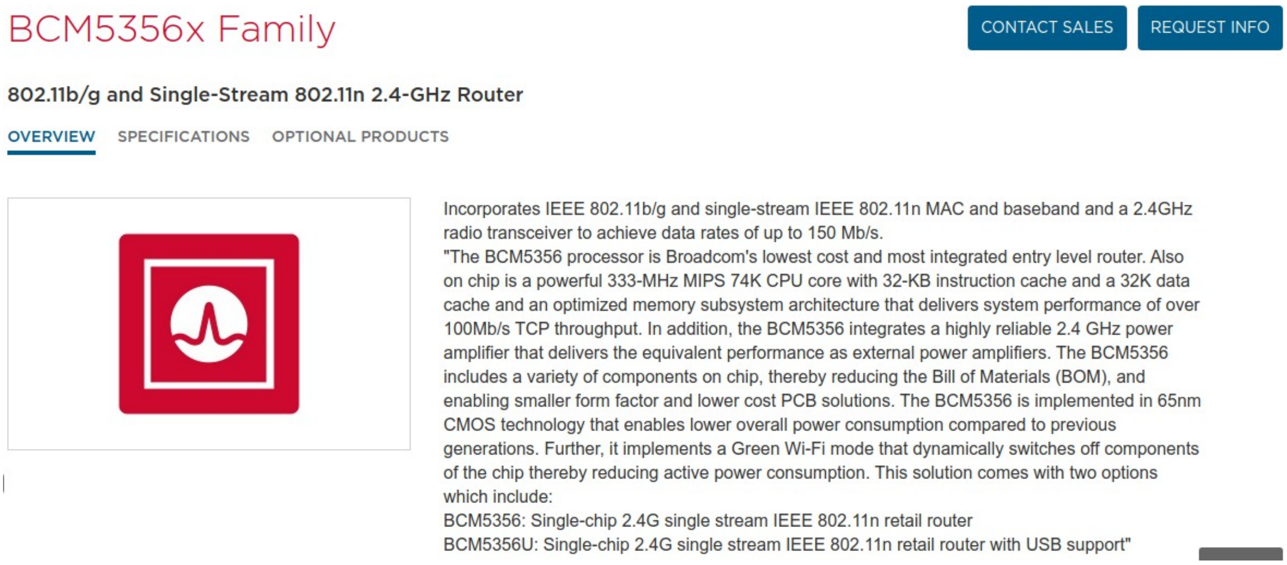


Fig 11. Broacom IC

- Let’s google the part number
  [Datasheet]



BCM5356x Family

802.11b/g and Single-Stream 802.11n 2.4-GHz Router

Incorporates IEEE 802.11b/g and single-stream IEEE 802.11n MAC and baseband and a 2.4GHz radio transceiver to achieve data rates of up to 150 Mb/s.
"The BCM5356 processor is Broadcom's lowest cost and most integrated entry level router. Also on chip is a powerful 333-MHz MIPS 74K CPU core with 32-KB instruction cache and a 32K data cache and an optimized memory subsystem architecture that delivers system performance of over 100Mb/s TCP throughput. In addition, the BCM5356 integrates a highly reliable 2.4 GHz power amplifier that delivers the equivalent performance as external power amplifiers. The BCM5356 includes a variety of components on chip, thereby reducing the Bill of Materials (BOM), and enabling smaller form factor and lower cost PCB solutions. The BCM5356 is implemented in 65nm CMOS technology that enables lower overall power consumption compared to previous generations. Further, it implements a Green Wi-Fi mode that dynamically switches off components of the chip thereby reducing active power consumption. This solution comes with two options which include:
BCM5356: Single-chip 2.4G single stream IEEE 802.11n retail router
BCM5356U: Single-chip 2.4G single stream IEEE 802.11n retail router with USB support"

Fig 12. Datasheet

- BCM5356 is a Baseband Micro-processor with 2.4 GHz radio transceiver. For more information about this chip check given link.

7.2.3 FR9886

- FR9886 is manufactured by fitipower integrated technology

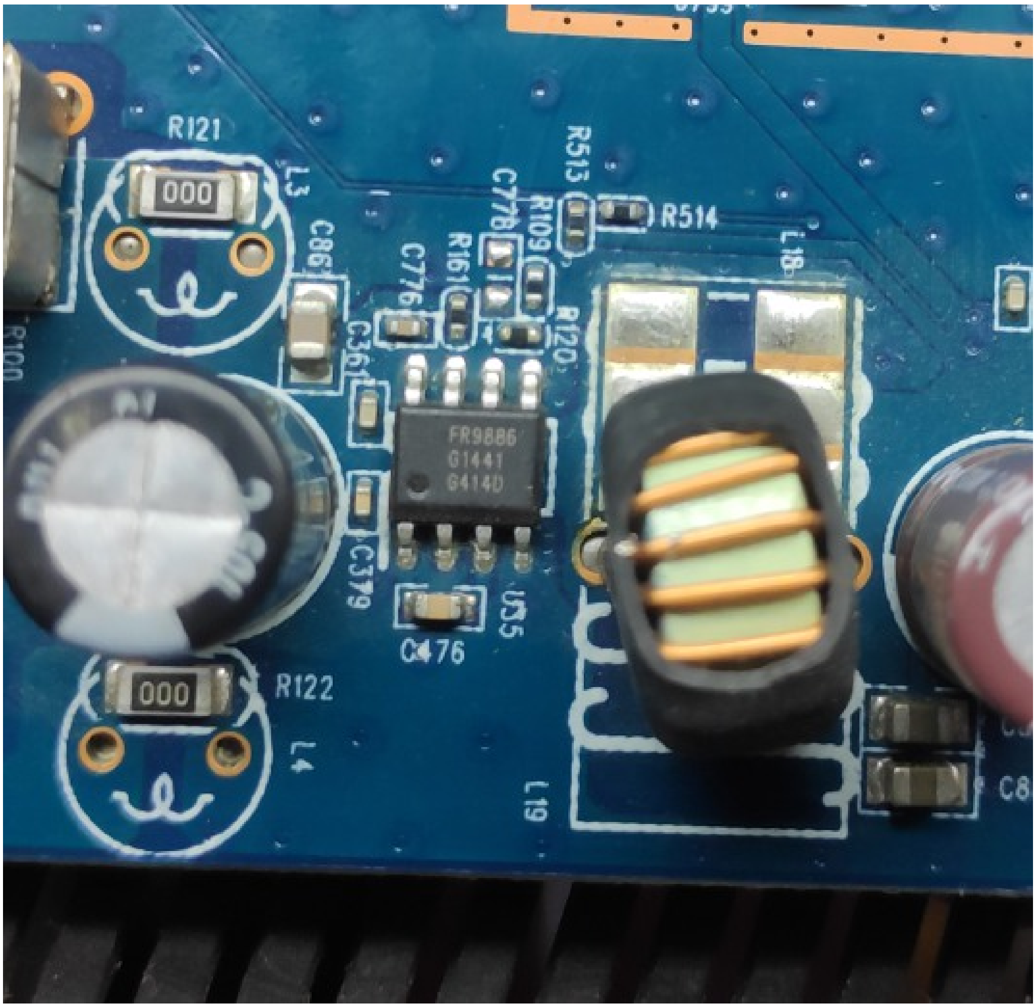
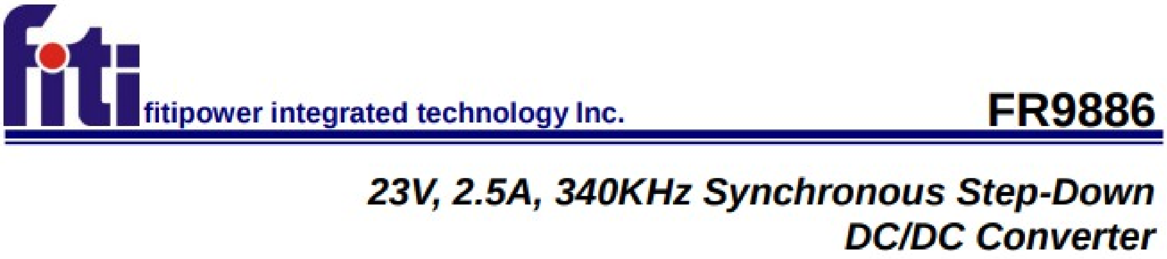

R121
000
L3
C86
R100
C778
R513
R514
R109
R161
C776
R120
L18
C361
FR9886
G1441
G414D
C379
C476
U35
R122
000
L4
L19
C8

Fig 13. Power IC

- Let’s google the part number
  [Datasheet]()



fiti fitipower integrated technology Inc.

FR9886

23V, 2.5A, 340KHz Synchronous Step-Down DC/DC Converter

Fig 14. Datasheet

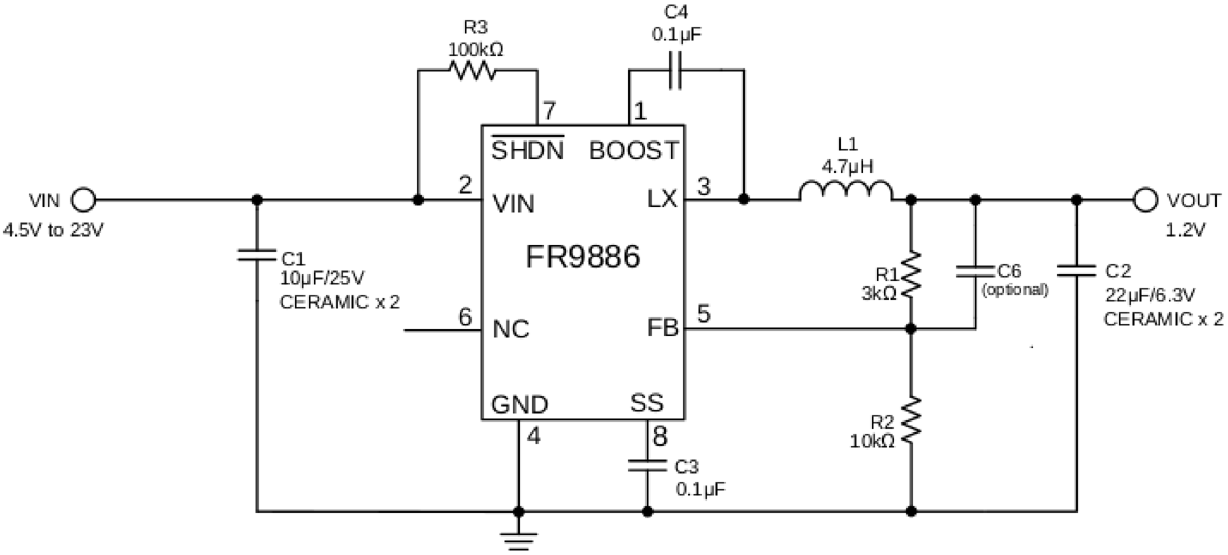

R3
100kΩ
C4
0.1µF
7
1
SHDN
BOOST
2
VIN
LX
3
L1
4.7µH
VIN
4.5V to 23V
VOUT
1.2V
FR9886
C1
10µF/25V
CERAMIC x 2
R1
3kΩ
C6
(optional)
C2
22µF/6.3V
CERAMIC x 2
6
NC
FB
5
GND
SS
4
8
C3
0.1µF
R2
10kΩ

Fig 15. Circuit Diagram

- FR9886 is a synchronous step-down DC/DC convertor
- This is the typical schematic diagram of fr9886 step down dc-dc convertor
- The input range is in between 4.5V to 23v DC and we will get the fixed output that is 1.25V

7.2.4 MXIC

- The IC is manufactured by Macronix.
- The part number is 25l3206e.

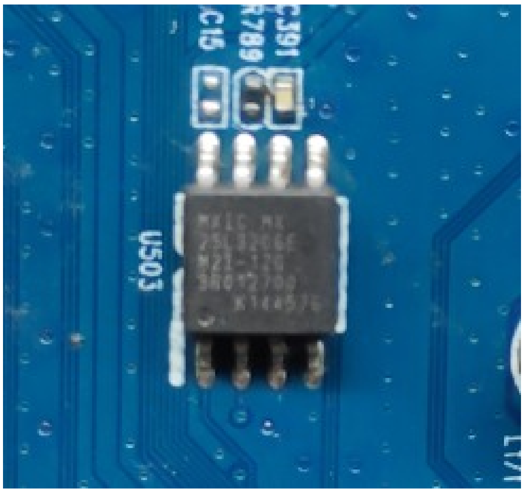
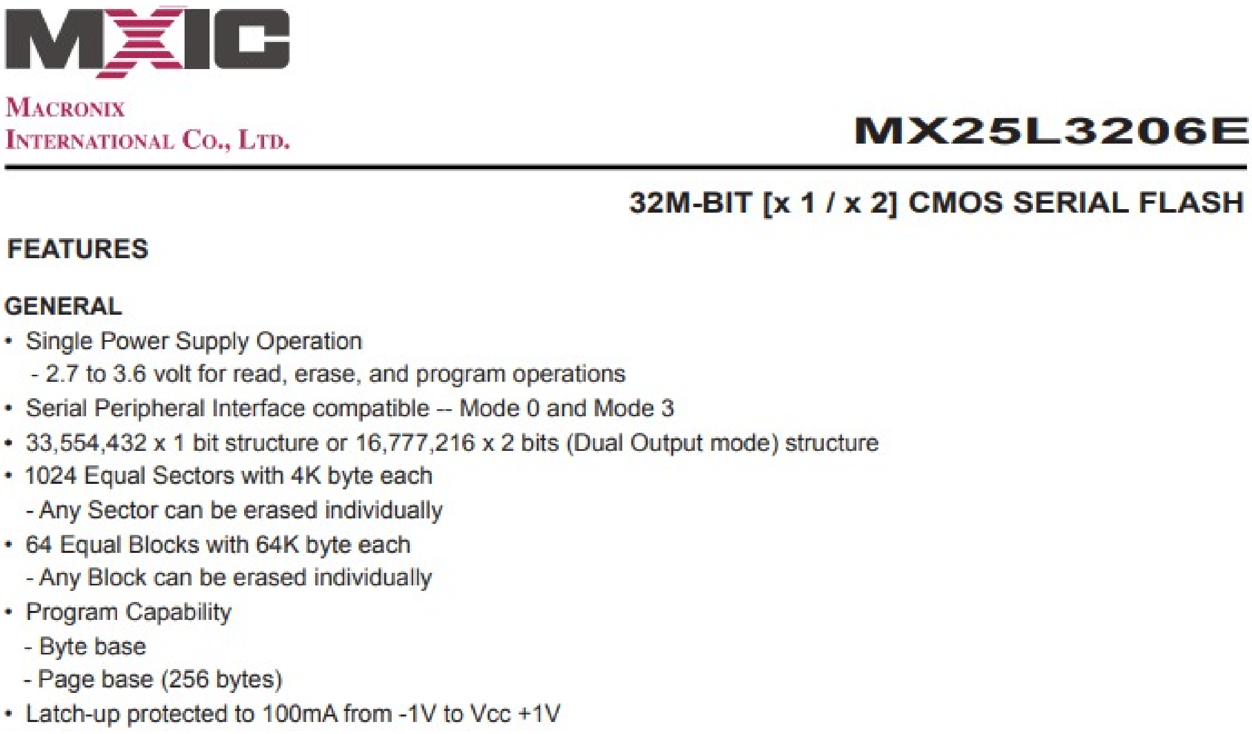

C391
R789
C15
U503
MXIC MX
25L3206E

Fig 16. Memory IC

- Let's download the datasheet and we know what this part is.
[Datasheet]



MACRONIX INTERNATIONAL CO., LTD.

MX25L3206E

32M-BIT [x 1 / x 2] CMOS SERIAL FLASH

FEATURES

GENERAL

- Single Power Supply Operation
  - 2.7 to 3.6 volt for read, erase, and program operations
- Serial Peripheral Interface compatible -- Mode 0 and Mode 3
- 33,554,432 x 1 bit structure or 16,777,216 x 2 bits (Dual Output mode) structure
- 1024 Equal Sectors with 4K byte each
  - Any Sector can be erased individually
- 64 Equal Blocks with 64K byte each
  - Any Block can be erased individually
- Program Capability
  - Byte base
  - Page base (256 bytes)
- Latch-up protected to 100mA from -1V to Vcc +1V

Fig 17. Datasheet

- This IC is an 8-Mb serial flash.
- This chip may store Linux based firmware for the target device.
- Pin Identification:
  This IC comes in different packages as shown below.
  In our case, the IC package is 8-LAND WSON (6*5MM).

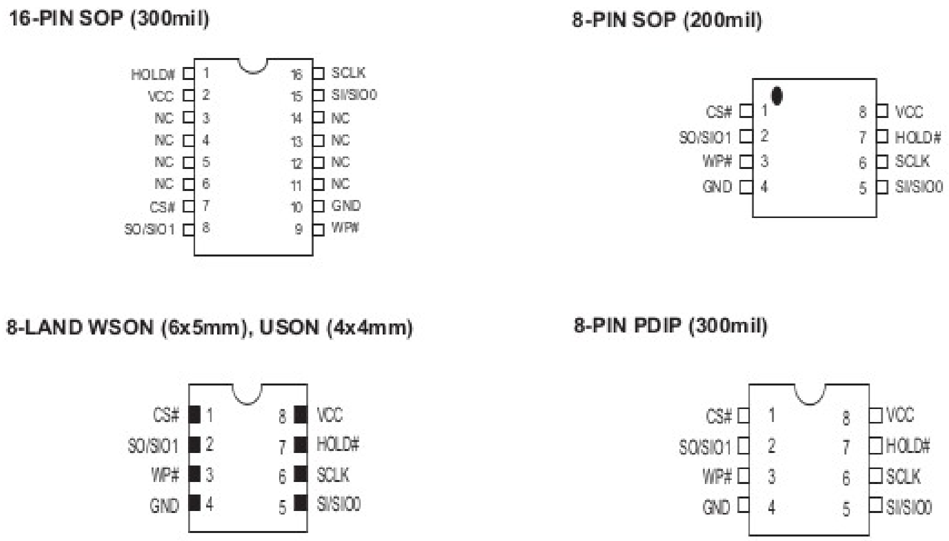
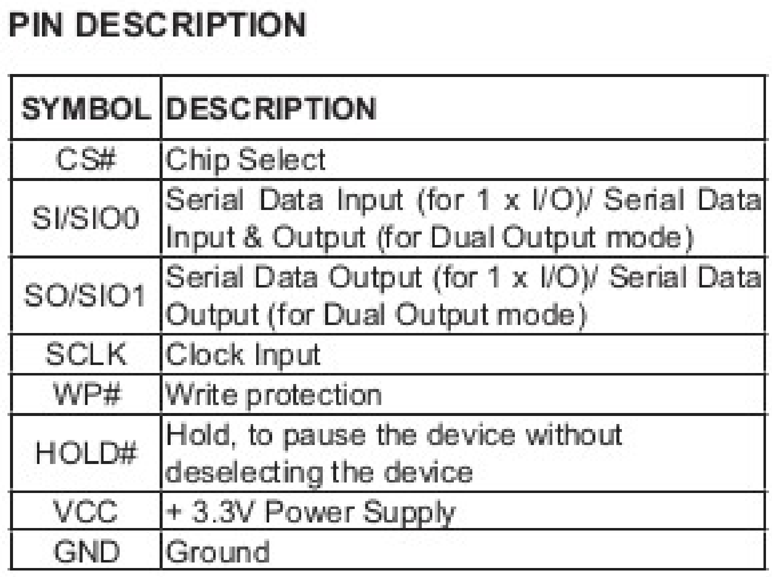

16-PIN SOP (300mil)
HOLD# 1
VCC 2
NC 3
NC 4
NC 5
NC 6
CS# 7
SO/SIO1 8
16 SCLK
15 SI/SIO0
14 NC
13 NC
12 NC
11 NC
10 GND
9 WP#
8-PIN SOP (200mil)
CS# 1
SO/SIO1 2
WP# 3
GND 4
8 VCC
7 HOLD#
6 SCLK
5 SI/SIO0
8-LAND WSON (6x5mm), USON (4x4mm)
CS# 1
SO/SIO1 2
WP# 3
GND 4
8 VCC
7 HOLD#
6 SCLK
5 SI/SIO0
8-PIN PDIP (300mil)
CS# 1
SO/SIO1 2
WP# 3
GND 4
8 VCC
7 HOLD#
6 SCLK
5 SI/SIO0

Fig 18. IC Package

- Pin description
  Here is the pin description taken from the datasheet. It is useful for making a connection with the hardware tool for reading and writing the firmware. This chip may store Linux based firmware for the target device.



PIN DESCRIPTION

SYMBOL	DESCRIPTION
CS#	Chip Select
SI/SIO0	Serial Data Input (for 1 x I/O)/ Serial Data Input & Output (for Dual Output mode)
SO/SIO1	Serial Data Output (for 1 x I/O)/ Serial Data Output (for Dual Output mode)
SCLK	Clock Input
WP#	Write protection
HOLD#	Hold, to pause the device without deselecting the device
VCC	+ 3.3V Power Supply
GND	Ground

Fig 19. IC Pinout

8. Serial Flash Recon

Out of the four components identified, let's look at the serial flash and how to extract data from the chip.

- Remove serial flash using soldering iron or hot air station from the router PCB.
- While desoldering, DO NOT heat it too much as it may damage the chip
- We need a PCB adapter for soldering removed ICs, as we can see in the image below.
- Be careful when soldering the IC; check the round notch that is PIN 1.

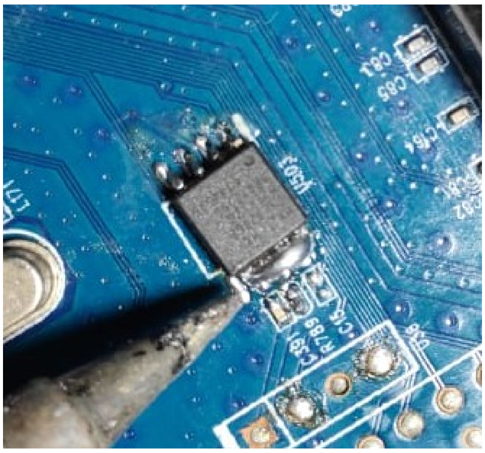


Fig 20. IC desolder

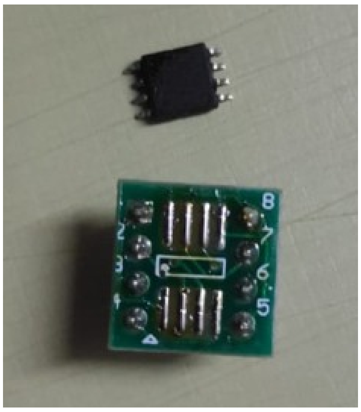


Fig 21. IC Adapter

8.1 Interfacing IC with our computer

To read or write the IC, we need can use commercial or open-source software. However, we need to interface the IC with our PC for the software to be able to communicate with the IC. We can use the EXPLIoT-Nano (https://expliot.io/products/expliot-nano) or any other hardware connector that speaks SPI protocol for reading and writing firmware from Serial flash.

- EXPLIoT-Nano is a compact hacker-friendly multi-purpose, multi-protocol hardware tool mainly used to debug and program microcontrollers/processors and flash chips.
- EXPLIoT-Nano can be configured to support hardware protocols including, UART, I2C, SPI, ARM SWD, and JTAG. Even though it runs on 3.3V, all I/O pins are 5V tolerant.
- After soldering the IC on the PCB adapter, make the connection properly, as shown below.
- Now you can extract the data (firmware in our case) and further analyze the information in the firmware. The chapter "Hardware Attack Surface: SPI" will cover this in more detail.

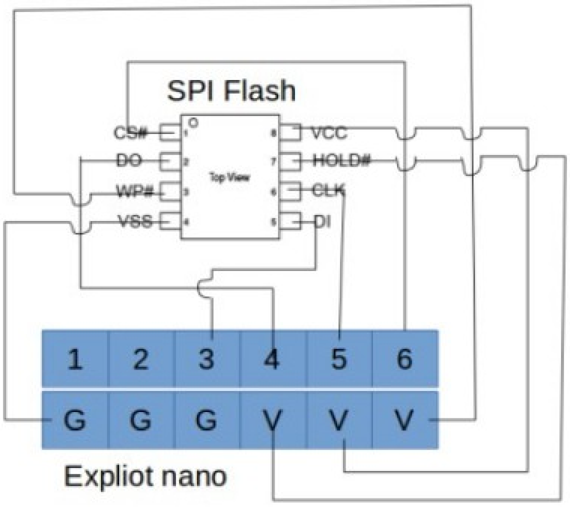


Fig 22. Connection Diagram

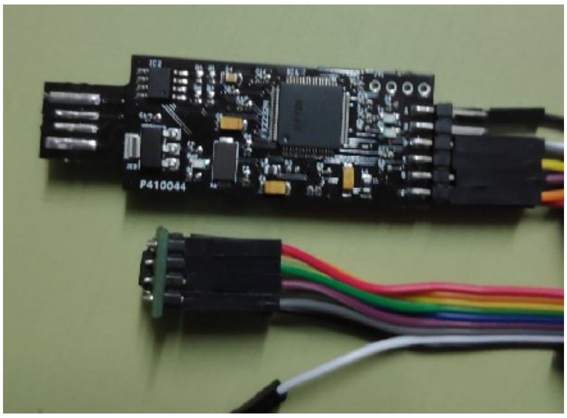


Fig 23. EXPLIoT Connection

9. Pinouts (headers)

Device manufacturers need a way to program the device with their firmware, so they generally use debug headers or test points to either debug or flashing firmware or recovering bricked devices.

As we have seen in this router, a PCB pin header has many pin headers. It's very important to know about what these headers contain. It may be a UART port, JTAG, SWD, anything that can directly talk with the main chip. Rarely these pin headers are labelled. In the next chapters, we will see what are the different debug ports, how to identify debug ports, different methods of identifying debug ports and its use cases.

10. Conclusion

We hope this chapter gave you an overview of the basics of hardware recon, how PCB analysis, component identification, its datasheets give you useful information to perform hardware assessment. This is a very basic overview of what you can and should do. In the upcoming chapter, we will cover more details of analyzing the different microcontrollers, memory chips, and hardware protocols commonly used in devices.

Chapter 14: Introduction to and Identification of Hardware Debug Ports

In the previous chapter, we discussed how to perform reconnaissance on hardware. In this chapter, we will discuss Hardware debug ports, what are the different debug ports, and how to find debug ports using different methods. Finding the hardware debug port process consists of several steps like looking at the various test pads, pinouts, headers. Rarely are these pins labeled on PCB, so we have to identify pins manually. Information from the microcontroller datasheet about pins and simple continuity check using a multimeter is good enough for IC packages like DIP/TSOP, TQF, but for packages like QFN, BGA, etc., where pins are not accessible on PCB, it is difficult to identify manually. For such cases where continuity check is not possible, tools like Bus Auditor or Jtagulator come in very handy. Once JTAG/SWD, UART, I2C pins are identified on PCB, access to the microcontroller, firmware extraction, root shell access is possible.

1. Why do PCBs have to debug ports?

The hardware debug ports can directly talk to the microcontroller. JTAG/SWD ports are used during development to download, debug firmware source code, recover bricked hardware, perform boundary-scan. A boundary scan is a method for testing interconnections on a printed circuit board (PCB). It’s the easiest way to troubleshoot the device. On the other hand, UART ports are typically used to access the device via a console for debugging the application running on the hardware. In this chapter, we are going to find UART, JTAG/SWD, I2C on DIVA Board using Bus Auditor.

2. Bus Auditor

Bus Auditor (https://expliot.io/products/bus-auditor-pre-order) is a compact multi-protocol tool used to scan and identify debug ports and communication interfaces exposed on any hardware board. It can brute force several hardware protocols including JTAG, Arm SWD, UART, I2C. This device has 16 independent channels to connect to the header pins on the target PCB. The inbuilt USB port is used for interfacing with the EXPLIoT framework (Internet of Things exploitation framework - https://gitlab.com/expliot_framework/expliot) running on the pc.

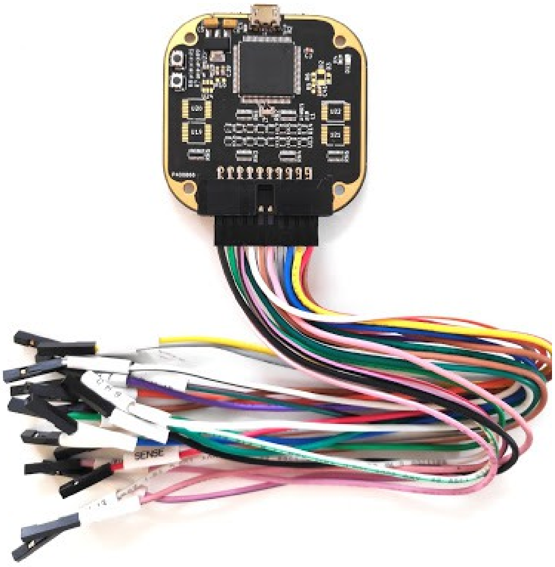


Fig 1. Bus Auditor

2.1 Features

- USB 2.0 high-speed interface.
- IEEE 1149.1 JTAG, and arm SWD support.
- UART RX, TX pins and Baud rate detection.
- i2c pin and i2c address detection.
- Adjustable target voltage level 1.2V-3.3V.
- 16 I/O channels with level transition and input protection.

So, let's have a look at the types of debug ports and identify it using Bus Auditor.

Note:
16 IO channels are labeled as CH0 to CH15 for easy identification.

3. DIVA

DIVA (Damn Insecure and Vulnerable application) IoT Board (https://expliot.io/products/diva-iot-board) is one of the EXPLIoT products. DIVA Board is a connected IoT device and a vulnerable target board designed to teach the basics of IoT security.

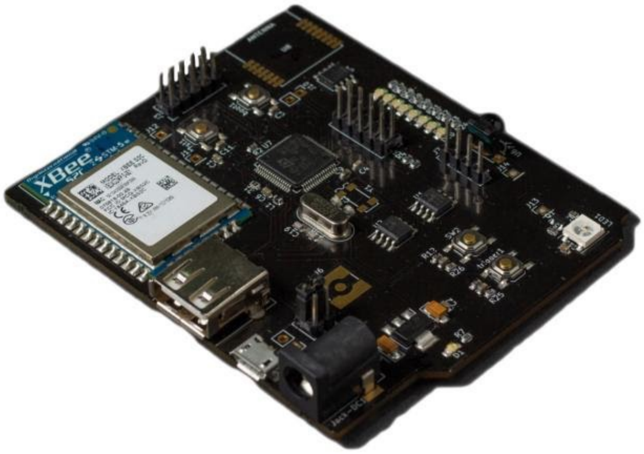


Fig 2. DIVA

The board provides a standard JTAG debug interface as well as an SWD interface that can be used as a debug port for firmware debugging and a UART port for a custom console for debug messages on the terminal.
There are 3 breakout headers on the DIVA board. We will scan the breakouts and find the debug ports.

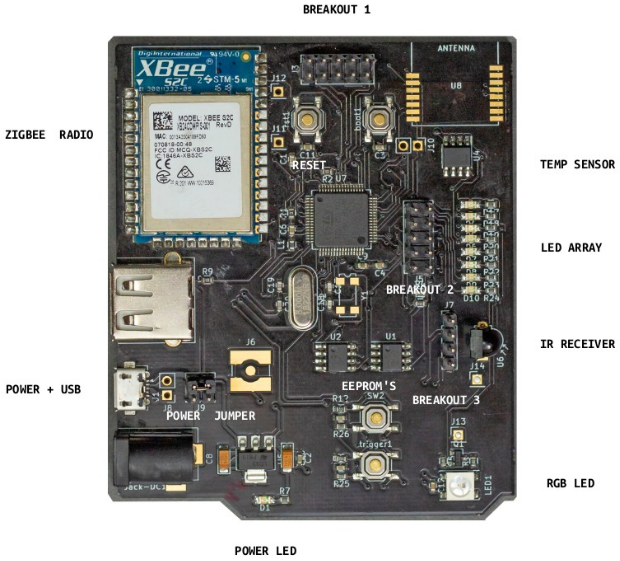

BREAKOUT 1
ZIGBEE RADIO
TEMP SENSOR
LED ARRAY
IR RECEIVER
POWER + USB
RGB LED
POWER LED
RESET
BREAKOUT 2
EEPROM'S
BREAKOUT 3
POWER JUMPER
ANTENNA

Fig 3. DIVA Board Layout

We can see the DIVA board layout in the photo with all peripherals and breakouts marked. We will identify JTAG/SWD, UART, I2C ports on breakout1, breakout2, and breakout3.

4. JTAG (Joint Test Action Group)

Nowadays, PCBs and SOCs have become smaller and advanced. Over time PCBs have complex and difficult to debug and program after production. To make it easy for testing, debugging, and programming the boards post-production, the industry developed an interface called Joint Test Action Group(JTAG). It is used for testing, debugging, and programming interfaces using a pinout (Header) or test pads on the PCB board.

Later this interface was adopted as IEEE 1149.1 standard. This standard defines the test access port (TAP) controller logic used in the processor with the JTAG interface. JTAG interface requires five pins. In many systems, the optional TRST pin is not implemented, so TRST is optional. resulting in its four-wire interface, the jtag pins are,

- TMS – Test Mode Select
- TCK – Test Clock
- TDI – Test Data In
- TDO – Test Data Out
- TRST – Test Reset (optional pin)

The chapter “Hardware Attack Surface- JTAG, SWD” will cover this in more detail

4.1 Identify JTAG Port on DIVA

We will scan breakout 1 or J3 header for possible JTAG interface. As of now, we don't know which pin is what for the JTAG port.

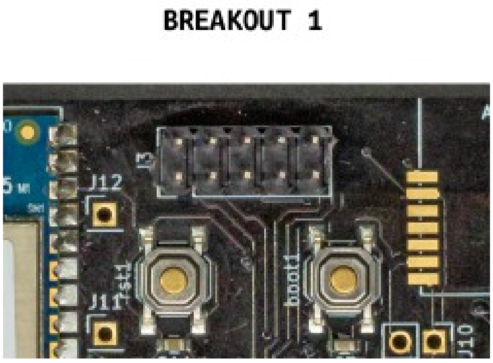


Fig 4. Breakout 1

1) Identify GND (Ground pin).
   The DIVA board should be powered off when checking continuity. Identify any metallic sheet area (usually metallic sheet areas are connected to the ground on the board) or identify the GND pin of the input DC supply of the DIVA board.

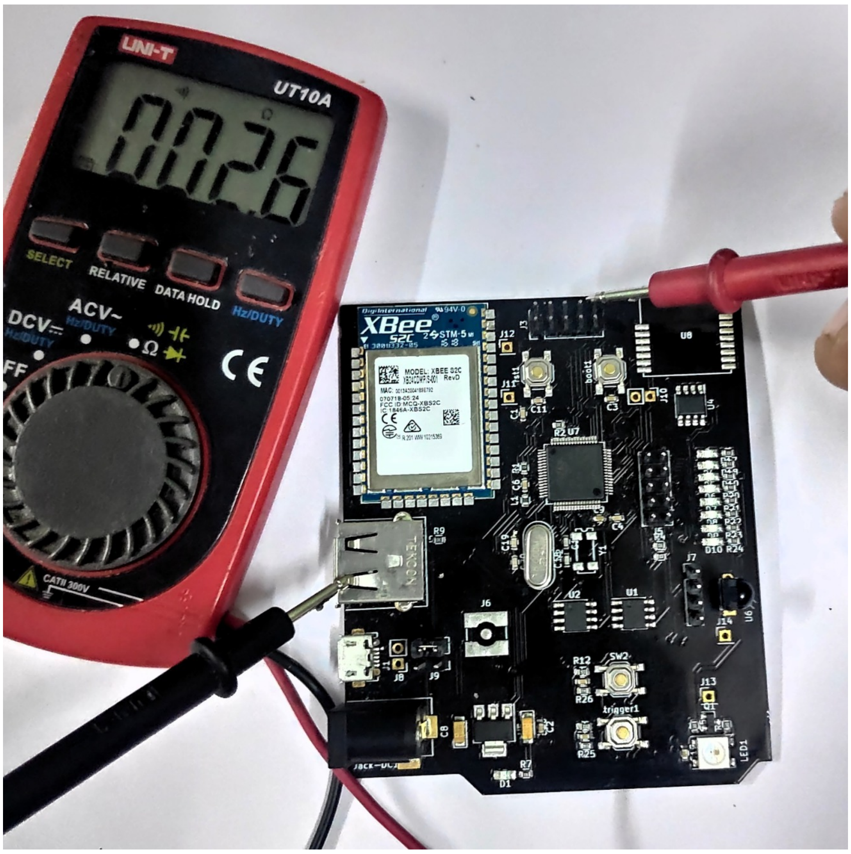


Fig 5. Identify ground pin

2) If the Multimeter beeps, it means that the pinout is connected to the ground i.e. it is a GND pin.Now let's connect Bus Auditor GND pin to diva GND pin and CH0-CH8 pins to header Breakout 1 (Bus Auditor pins are labeled from 0 to 15)

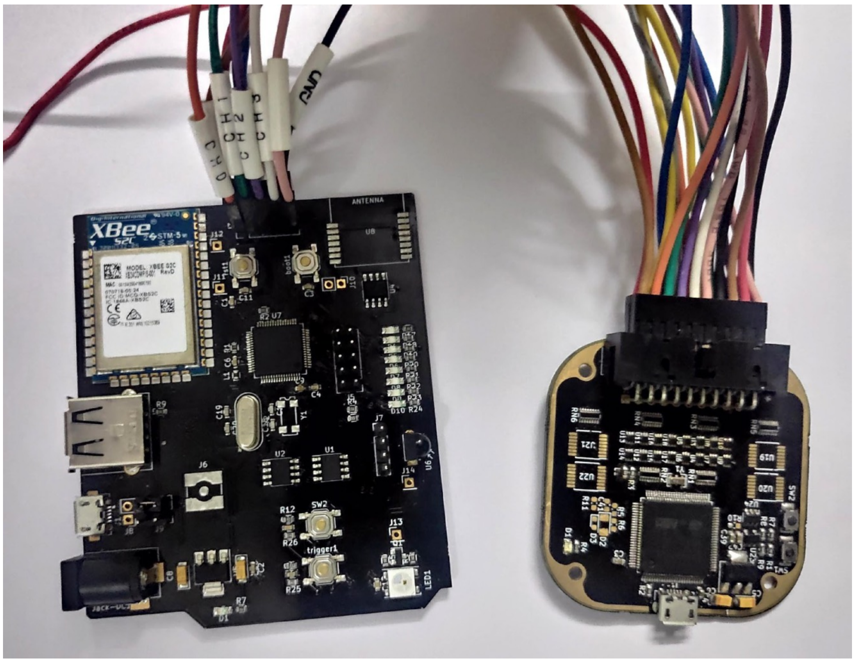


Fig 7. Pin Connections

3) Connect both devices to pc using USB to Micro USB cable.

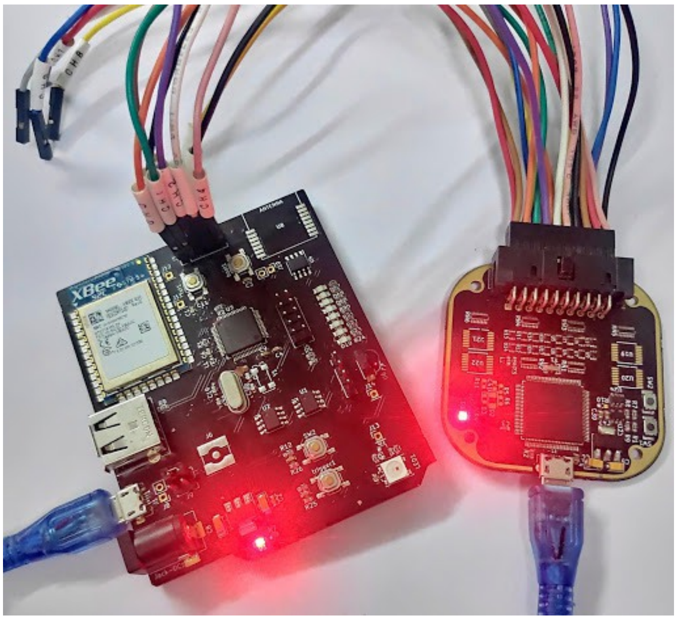


Fig 9. Boards Power up

4) Run the EXPLIoT framework (use root/sudo if your system does not allow user processes to connect to tty* devices).
   - `$ expliot`

5) Run the Busauditor plugin
   - `ef> run busauditor.generic.jtagscan -v 3.3 -p /dev/ttyACM0 -s 0 -e 10`

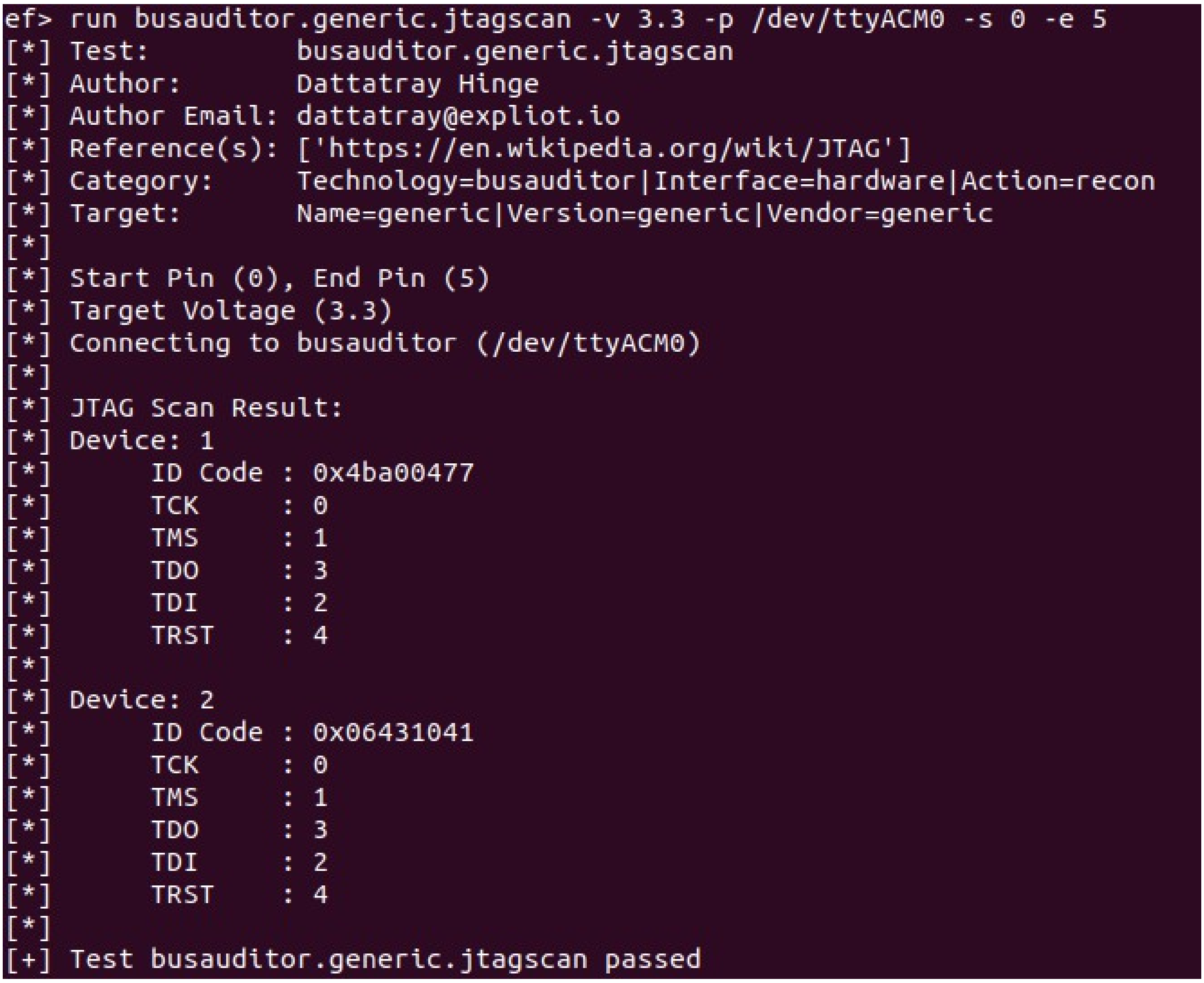
-v is for setting Voltage.

-p dev/tty* port.

-s starting channel.

-e ending channel.

wait for the plugin to complete JTAG port scanning.



```
ef> run busauditor.generic.jtagscan -v 3.3 -p /dev/ttyACM0 -s 0 -e 5
[*] Test:          busauditor.generic.jtagscan
[*] Author:        Dattatray Hinge
[*] Author Email: dattatray@expliot.io
[*] Reference(s): ['https://en.wikipedia.org/wiki/JTAG']
[*] Category:      Technology=busauditor|Interface=hardware|Action=recon
[*] Target:        Name=generic|Version=generic|Vendor=generic
[*]
[*] Start Pin (0), End Pin (5)
[*] Target Voltage (3.3)
[*] Connecting to busauditor (/dev/ttyACM0)
[*]
[*] JTAG Scan Result:
[*] Device: 1
[*]       ID Code : 0x4ba00477
[*]       TCK     : 0
[*]       TMS     : 1
[*]       TDO     : 3
[*]       TDI     : 2
[*]       TRST    : 4
[*]
[*] Device: 2
[*]       ID Code : 0x06431041
[*]       TCK     : 0
[*]       TMS     : 1
[*]       TDO     : 3
[*]       TDI     : 2
[*]       TRST    : 4
[*]
[+] Test busauditor.generic.jtagscan passed
```

Fig 9. JTAG Scan

4.2 Bus Auditor Result

Here we found JTAG pins and ID code
TCK: Ch0
TMS: Ch1
Fig 10. SWD Scan
TDO: Ch3
TDI: Ch2
TRST: Ch4
and ID CODEs are
ID Code : 0x4ba00477
ID Code : 0x06431041

ID CODE is a 32bit number unique manufacturer ID that identifies the part type in the JTAG chain.The ID code differs from vendor to vendor. Here we found 2 ID codes one is from the main controller from STM electronics, and one is the jtag controller in the SOC.

5. SWD (Serial Wire Debugger)

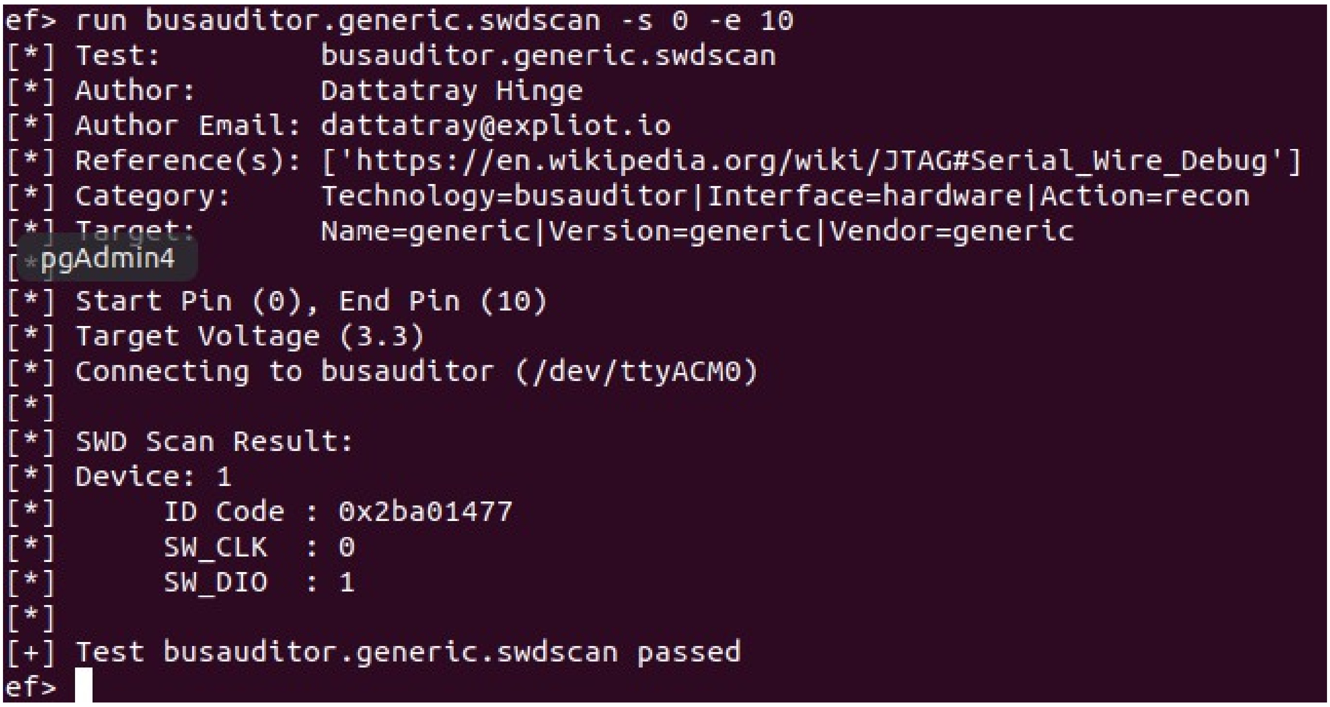
The JTAG connector requires four pins for many applications. A variant of the JTAG was introduced called serial wire debug (SWD). It uses just 2 pins with clock and bi-directional data pin SWDIO and SWCLK are overlaid on the TMS and TCK pins.

5.1 Identify SWD Port on DIVA

We will keep the connection the same for identifying SWD pins,
Run command:

- `ef> run busauditor.generic.swdscan -s 0 -e 10`

(Note: voltage and device port it will take default 3.3v and ttyACM0)



```
ef> run busauditor.generic.swdscan -s 0 -e 10
[*] Test:         busauditor.generic.swdscan
[*] Author:       Dattatray Hinge
[*] Author Email: dattatray@expliot.io
[*] Reference(s): ['https://en.wikipedia.org/wiki/JTAG#Serial_Wire_Debug']
[*] Category:     Technology=busauditor|Interface=hardware|Action=recon
[*] Target:       Name=generic|Version=generic|Vendor=generic
[ *pgAdmin4
[*] Start Pin (0), End Pin (10)
[*] Target Voltage (3.3)
[*] Connecting to busauditor (/dev/ttyACM0)
[*]
[*] SWD Scan Result:
[*] Device: 1
[*]      ID Code : 0x2ba01477
[*]      SW_CLK  : 0
[*]      SW_DIO  : 1
[*]
[+] Test busauditor.generic.swdscan passed
ef> 
```

Fig 10. SWD Scan

5.2 Bus Auditor Result

Here we found Pins
SW_CLK: CH0
SW_DIO: CH1
Fig 11. I2C-Communication
Fig 12. Breakout 2
and ID Code : 0x2ba01477

We found SWD pins and ID code. Using this information, attackers can extract firmware, reverse engineer the logic, and flash malicious firmware on the device. JTAG/SWD adds significant capability when debugging microcontrollers. However, they require access to the JTAG/SWD pins on the processor. Debugging tools like GDB and OpenOCD (Open On-Chip Debugger) can be used over JTAG/SWD to debug/manipulate the code/execution.

6. I2C(Inter-Integrated Circuit)

I2C is an asynchronous, multi-master, multi-slave, serial communication bus, which means that multiple chips can be connected to the same bus and each one can act as a master by initiating data transfer. I2C is used to communicate between low-speed peripherals ICs and processor/microcontroller within a short distance on the same board.

Like UART communication, I2C only uses two wires to transmit and receive data between devices.

- SDA (Serial Data) – The line for the master and slave to send and receive data.
- SCL (Serial Clock) – The line that carries the clock signal.

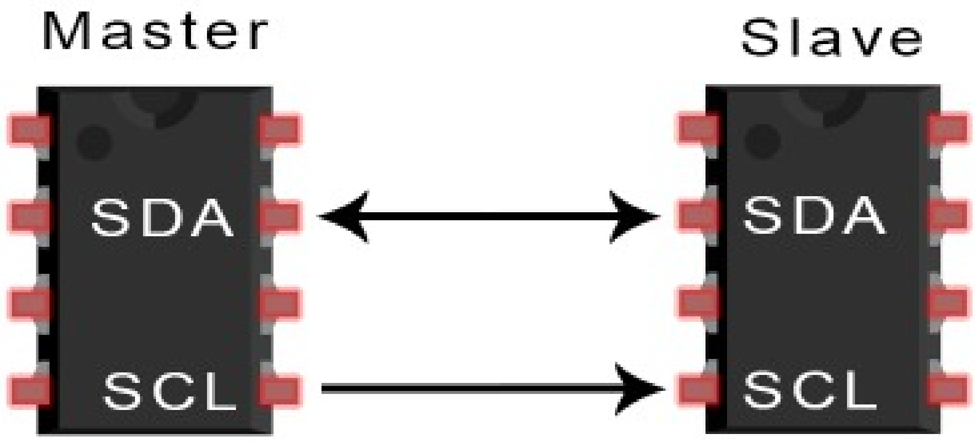


Fig 11. I2C-Communication
(Image source: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/)

6.1 Identify I2C on the DIVA board

We will connect the bus auditor to a DIVA pin header breakout 2

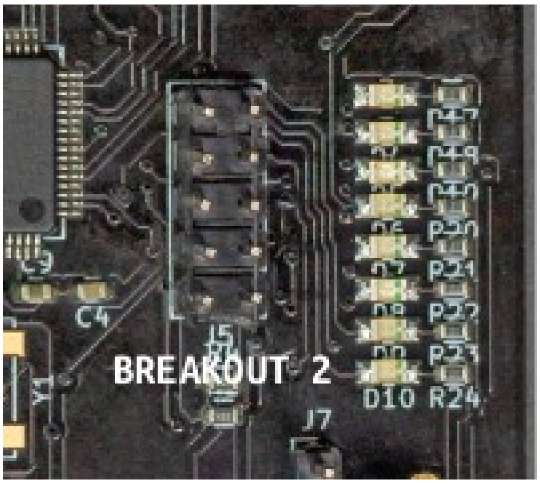


Fig 11. I2C-Communication

1) Identify the ground pin as the same we found earlier using a multimeter continuity test. Connect GND and other CHO-CH8 channels to the pin header breakout board see in the below figure connection made

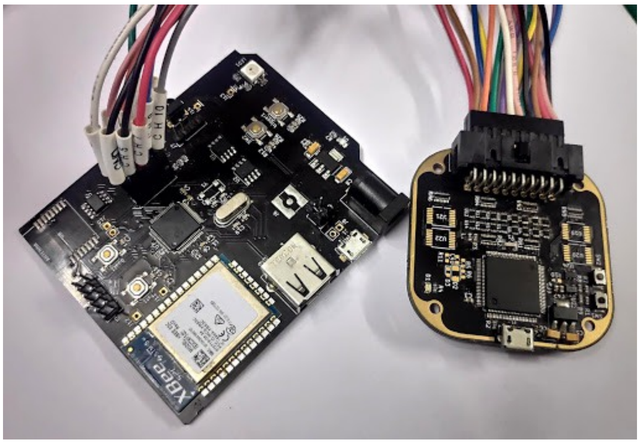


Fig 13. Pin Connections

2) Connect both devices to pc using USB to Micro USB cable.

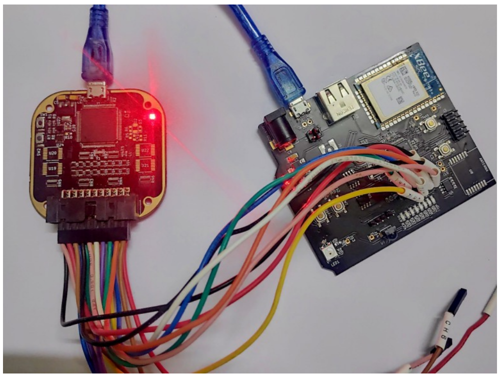
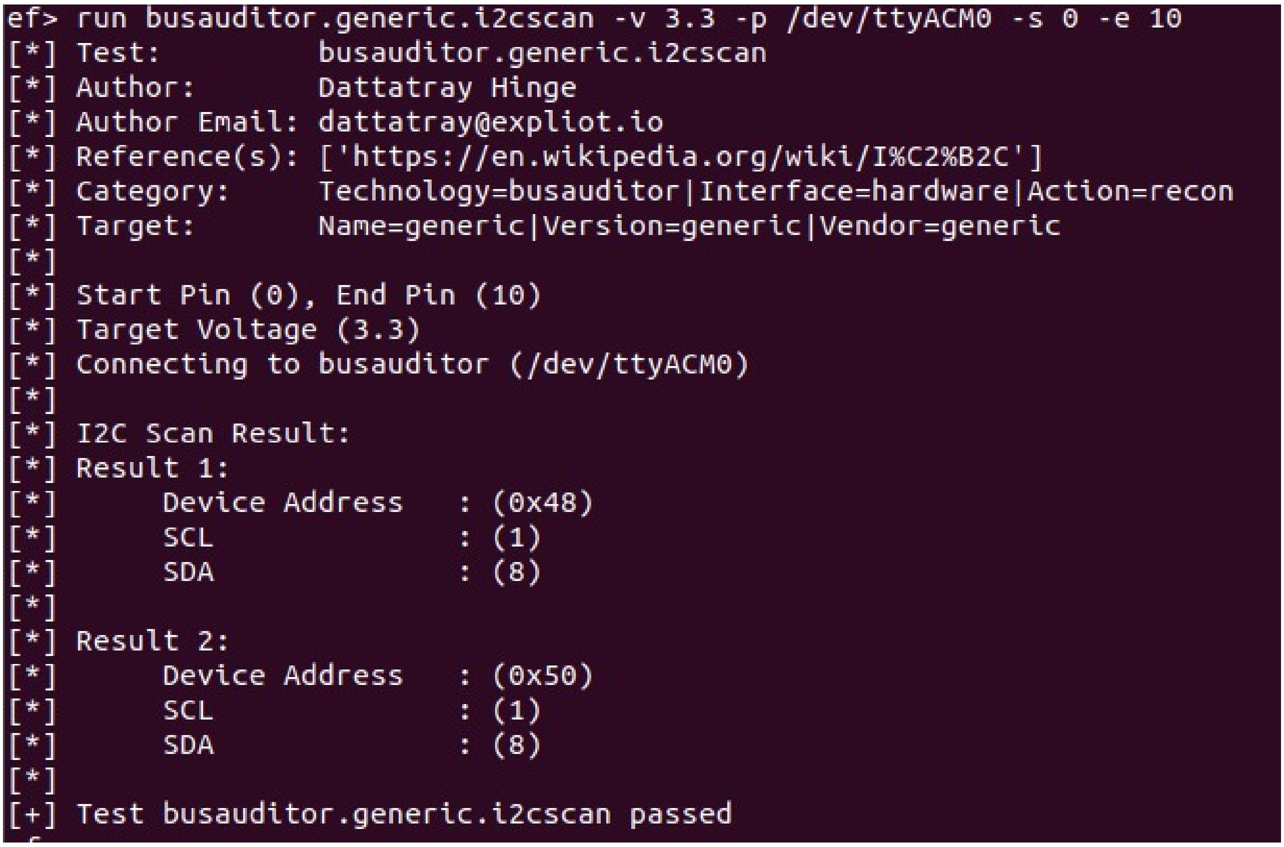


Fig 14. Boards Power up

3) Open EXPLIoT framework
Run command:

- `ef> run busauditor.generic.i2cscan-v 3.3 -p /dev/ttyACM0 -s 0 -e 10`



```
ef> run busauditor.generic.i2cscan -v 3.3 -p /dev/ttyACM0 -s 0 -e 10
[*] Test:         busauditor.generic.i2cscan
[*] Author:       Dattatray Hinge
[*] Author Email: dattatray@expliot.io
[*] Reference(s): ['https://en.wikipedia.org/wiki/I%C2%B2C']
[*] Category:     Technology=busauditor|Interface=hardware|Action=recon
[*] Target:       Name=generic|Version=generic|Vendor=generic
[*]
[*] Start Pin (0), End Pin (10)
[*] Target Voltage (3.3)
[*] Connecting to busauditor (/dev/ttyACM0)
[*]
[*] I2C Scan Result:
[*] Result 1:
[*]       Device Address    : (0x48)
[*]       SCL               : (1)
[*]       SDA               : (8)
[*]
[*] Result 2:
[*]       Device Address    : (0x50)
[*]       SCL               : (1)
[*]       SDA               : (8)
[*]
[+] Test busauditor.generic.i2cscan passed
```

Fig 15. I2C Scan

6.2 Bus Auditor Result

We can see that the bus auditor identified 2 pins out of 8 pins and found the device address.

SCL: CH1
SDA: CH8
Device Address : (0x48)
Device Address : (0x50)

One of the use cases of I2C is in EEPROM chips that are connected to the microcontroller I2C pins and typically store data or code. Typical attacks would include tampering with the data, extracting sensitive information, corrupting the data, etc. We should analyze the data at rest on the EEPROM chip as well as perform run-time analysis by sniffing the I2C communication to understand the behavior and security implications.

7. UART (Universal Asynchronous Receiver-Transmitter)

UART (Universal Asynchronous Receiver Transmitter) is a hardware component that allows asynchronous serial communication between two hardware components. This can be on the same devices (microcontrollers connected to any sensor on the device) or two different devices (device microcontroller talking to a PC via an external UART hardware)

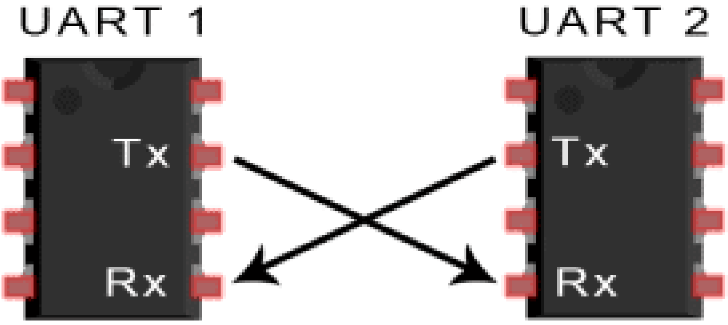


Fig 16. UART Communication
Image source : (https://www.circuitbasics.com/basics-uart-communication/)

7.1 Identify UART on DIVA

Here we will identify UART pinouts on the PCB board when we do not know the microcontroller UART pins and the datasheet of the microcontroller.

1) We will connect bus auditor channel pins to pin header breakout 3, which is on the diva.

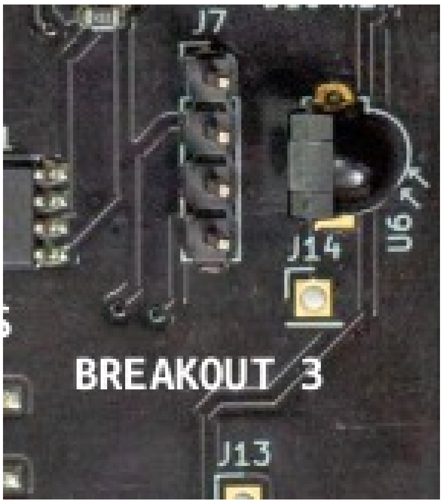


Fig 17. Breakout 3

2) First, we will identify the ground pin using a multimeter as explained earlier. One pin is VCC no need to connect it, and for identifying TX, RX we will connect Ch0 and CH1 of Bus Auditor.
3) Connect both devices to PC using USB to Micro USB cable.

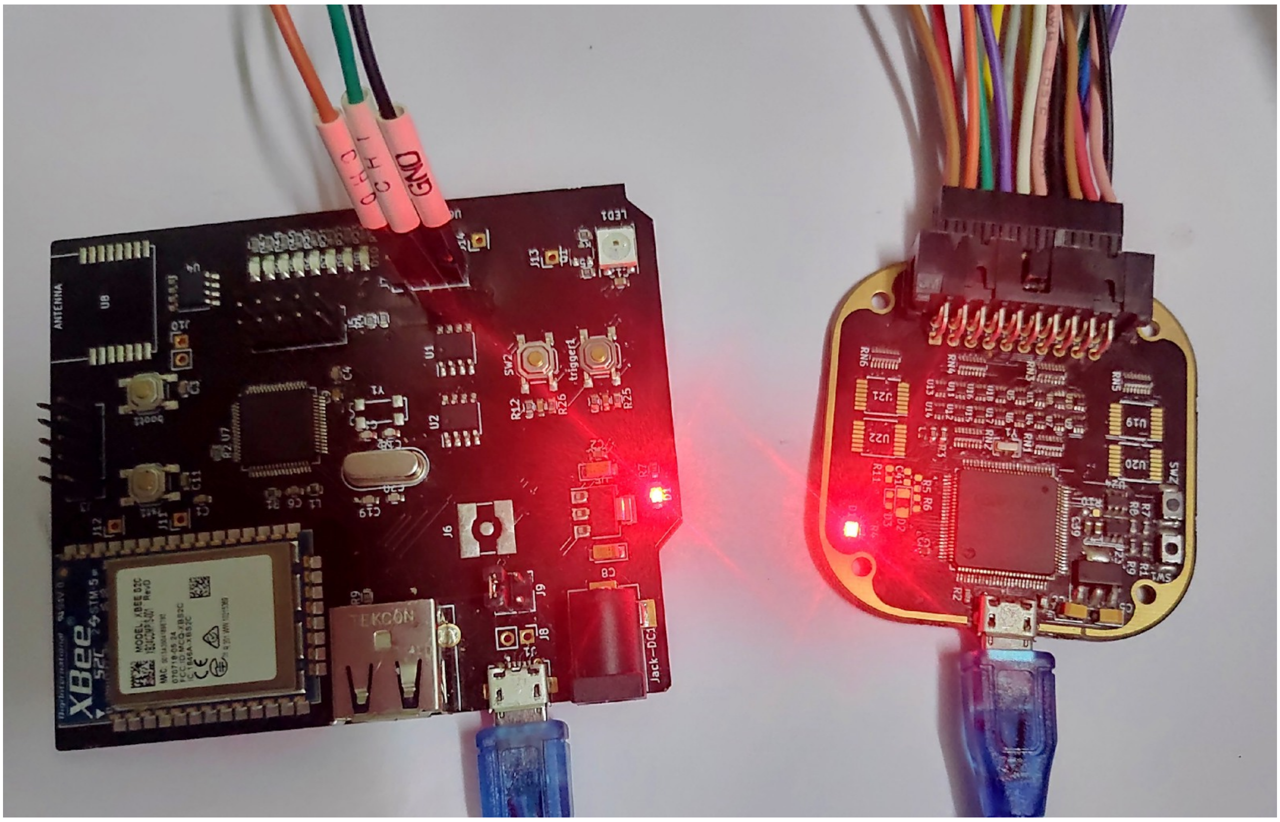
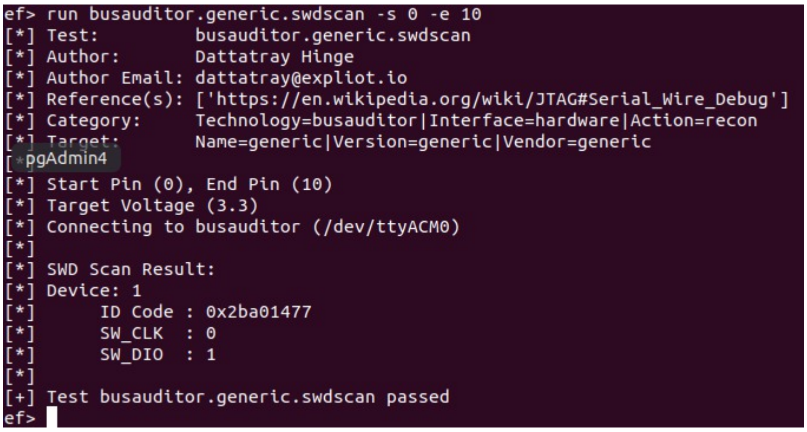


Fig 18. Pin Connections

4) Open the EXPLIoT framework

Run command:

- `ef> run busauditor.generic.uartscan -v 3.3 -p /dev/ttyACM0 -s 0 -e 1`



```
ef> run busauditor.generic.swdscan -s 0 -e 10
[*] Test:         busauditor.generic.swdscan
[*] Author:       Dattatray Hinge
[*] Author Email: dattatray@expliot.io
[*] Reference(s): ['https://en.wikipedia.org/wiki/JTAG#Serial_Wire_Debug']
[*] Category:     Technology=busauditor|Interface=hardware|Action=recon
[*] Target:       Name=generic|Version=generic|Vendor=generic
[ pgAdmin4
[*] Start Pin (0), End Pin (10)
[*] Target Voltage (3.3)
[*] Connecting to busauditor (/dev/ttyACM0)
[*]
[*] SWD Scan Result:
[*] Device: 1
[*]      ID Code : 0x2ba01477
[*]      SW_CLK  : 0
[*]      SW_DIO  : 1
[*]
[+] Test busauditor.generic.swdscan passed
ef> 
```

Fig 19. UART Scan

7.2 Bus auditor Result

We found the correct baud rate and correct RX, TX pins,

Baud rate is 9600

RX: CH1

TX: CH0

It is an interesting attack surface as it may allow read/write access to the application, running on the device, over serial. In many devices, UART ports on the board are left open, which anyone can connect and access over serial to get a console of some sort i.e. simple shell, custom command line consoles, log output, etc. A device will typically have a group of pin-outs connected to the microcontroller UART RX and TX pins, which are used for sending and receiving serial data. Here we identified the UART port using the Bus auditor and in the upcoming chapters, we will see how to access the UART ports on a device.

Bus Auditor

Bus Auditor is our very own home-grown debug port scanner. If you would like to purchase Bus Auditor, or, for that matter, any of our other hardware security tools, feel free to check out our online store at EXPLIoT - https://expliot.io/collections/frontpage

8. Conclusion

We hope this chapter gave you an overview of the hardware debug ports, how Bus auditor scans and identifies debugging and communication interfaces exposed on the PCB test pads and pinout headers, which gives you useful information to perform hardware assessment. In upcoming chapters, we will cover more details of each hardware debug port.

SPI
Chapter 15: Hardware Attack Surface - SPI

1. Introduction

An IoT product eco-system comprises a hardware device(s), firmware, network, communication protocols, web, mobile, cloud applications, and the eco-system is growing exponentially with growth in the IoT market.

It comprises of a wide range of attack surfaces. To get a better understanding of the IoT attack surface, you can refer to the 2nd Chapter of this e-book. An embedded device has many communication protocols at the hardware level that enable the microprocessor/microcontroller to communicate with its peripheral components.

This chapter will discuss one of the hardware communication protocols, **Serial Peripheral Interface (SPI)**. If you are a beginner at hardware hacking and want to get started at a basic level, stay tuned, you are in the right place.

SPI is a synchronous serial communication interface. It was designed primarily to communicate (transfer data) between the components located on the same PCB (Printed Circuit Board). It was developed by Motorola. It provides a full-duplex communication mode (i.e., sender and receiver can transmit and receive the data simultaneously). It has a master-slave architecture that supports a single master and multiple slaves. It is a 4-wire serial interface. It consists of the following four logic signals:

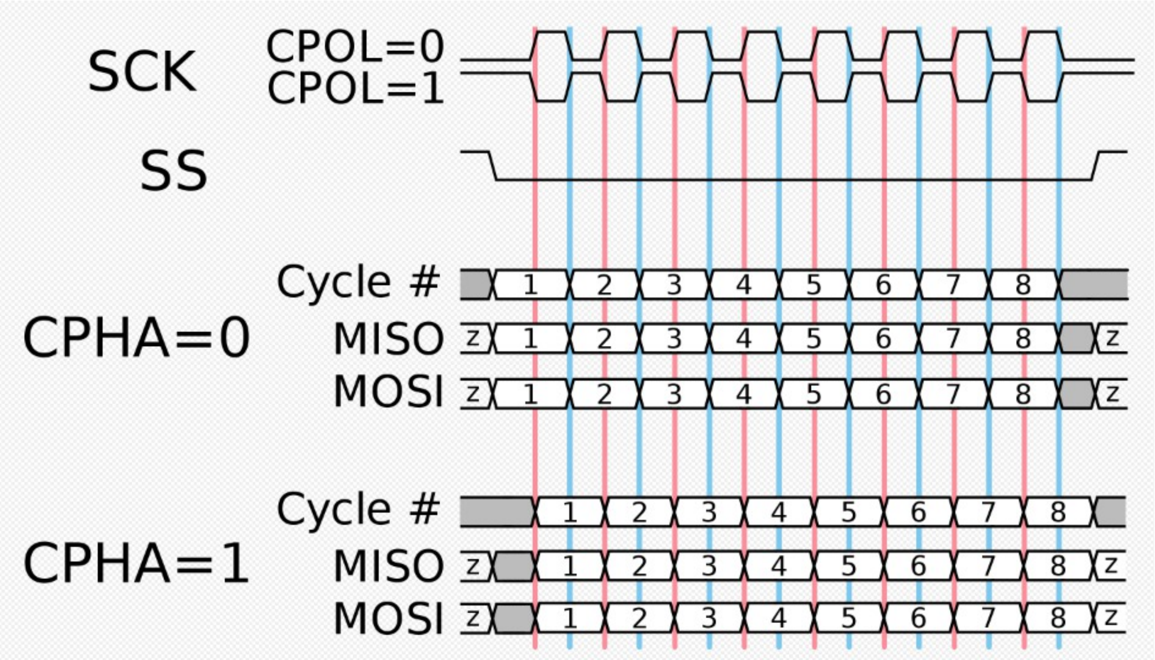
- MOSI (Master Output -> Slave Input) - Master outputs the data to the slave.
    - SDO, DO, DOUT, SO - These are different names for MOSI pins on the master device that connects to MOSI pins on the slave device.
    - SDI, DI, DIN, SI - These are different names for MOSI pins on the slave device that connects to MOSI pins on the master device.
- MISO (Master Input <- Slave Output) - Slave outputs the data to the master.
    - SDO, DO, DOUT, SO - These are different names for MISO pins on the slave device that connects to MISO pins on the master device.
    - SDI, DI, DIN, SI - These are different names for MISO pins on the master device that connects to MISO pins on the slave device.
- SS (Slave Select/Chip Select) - Output from master. Master uses this signal to select the slave device.
- SCLK (Serial Clock) - Output from the master. To start the communication with the slave, the master configures and sends the clock signal. It also provides the speed of data transfer between the master and the slave.

SPI supports 4 basic modes of operation:

SPI Modes	Clock Polarity (CPOL/CKP)	Clock Phase (CPHA/CKE)
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0
Mode 3	1	1

All the SPI devices do not support all these modes. Different SPI devices may either work on logic 0 or logic 1. They may have different data latching mechanism. For example, they may support latching data either at rising or falling edge; some may support either low or high as an idle state. So, having these different SPI modes provides the flexibility to different types of devices to perform SPI communication. The mode supported by any SPI device can be found in their respective datasheet.

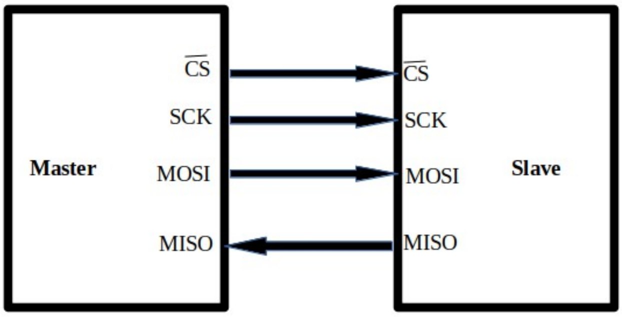
These modes operate similarly but have different timing relation with the SCK clock edge. The timing diagram showing different modes with clock polarity (CPOL/CKP) and clock phase(CPHA/CKE) is shown below. (Source - wikipedia page)



Depending on the mode that is to be used for communication, the master configures clock polarity and clock phase.

- If CPOL/CKP = 0 : Clock is idle at 0. The leading edge is a rising edge, and the trailing edge is falling.
- If CPOL/CKP = 1 : Clock is idle at 1. The leading edge is falling, and the trailing edge is a rising edge.
- If CPHA/CKE = 0: SDO shifts the data on the trailing edge of the preceding clock cycle, and SDI samples the data on the rising edge.
- If CPHA/CKE = 1: SDO shifts the data on the leading edge of the current clock cycle and SDI samples the data on the falling edge.

Moreover, the combination of CPOL and CPHA gives the different modes of SPI communication. The diagram below depicts the high-level SPI communication between the master and the slave.



1. Master selects the slave device by sending logic 0 on the CS signal. (CS is active low)
2. Master sends the clock signal
3. Master sends the data/command on the MOSI line, and the slave reads it
4. Slave sends the data/response on the MISO line, and the master reads it

There are two shift registers involved, of given word size, often 8 bits, although other word sizesare also common. One shift register is present in the master, and another is present in the slave. They form an inter-chip circular buffer. The most significant byte (MSB) of the data is usually shifted out first. Its detailed explanation is available on the wikipedia page.

2. Possible Attack Scenarios

In an embedded device, the SPI protocol is used to communicate with various peripherals on the device. It could be sensors, control devices, LCD, SD Card, EEPROM, Flash memory, etc. If an attacker gets access to the device hardware, there are possibilities that he may find the SPI device as an attack surface on the hardware.

- Sniffing the communication between an SPI device like sensors, controlling device, memory chip, etc. and the controller/processor. It may lead to the stealing/leak of sensitive information by the attacker. For example, in some of the embedded devices, an SPI EEPROM is used to store sensitive information, keys, logs, etc. This sensitive info can be leaked if an attacker sniffs the SPI communication between the EEPROM and the controller/processor. After getting this sensitive info, an attacker can damage in many ways and maybe on a large scale if the hardware vendor has used the same sensitive info like the same keys in all the hardware in the production.
- Extracting the firmware/sensitive info from SPI memory chips. Some embedded devices use SPI Flash memory to store firmware inside them. There are already tools/methods available to read NAND/NOR SPI/Parallel flash memories. The attacker can read the flash memory and extract the firmware out of it. The firmware can further be reverse engineered and analyzed for potential vulnerabilities or cloning. If the firmware is not protected, the IP can be easily stolen. If the chips hold configuration or other sensitive data (instead of firmware), extracting and analyzing can lead to unearthing new vulnerabilities, as suggested in the above point. Patching the firmware in the SPI flash memory chip. In this case, if the firmware is not securely managed on the device, an attacker can patch the firmware with a backdoor and reprogram the SPI flash memory with the malicious firmware.

3. Attacking hardware via SPI

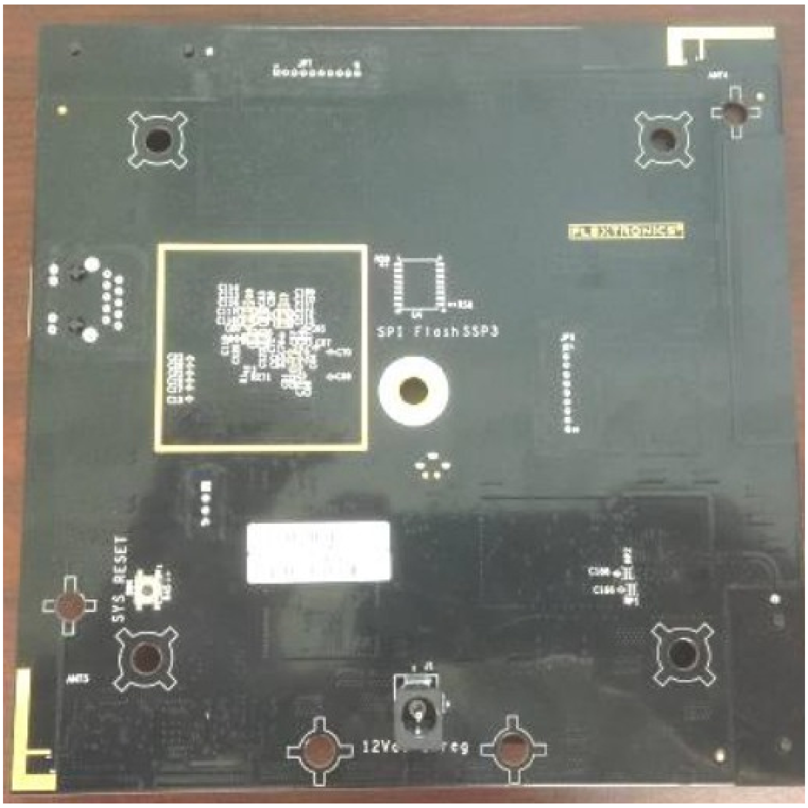
There are various ways in which the attacker can attack the hardware via SPI. Few methods to perform the attacks are explained in the below section.

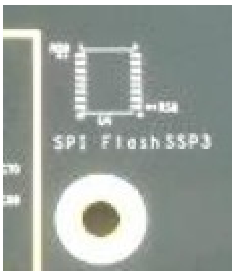
3.1 Recon

Case 1: You do not have the actual hardware with, but you know, the FCCID number of the device.

Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. Seeing the internal image, you may get the hint for an SPI chip on the hardware. Yayy!! Once you know that you got the attack surface, you can get/purchase that device and perform further required steps to attack the device.

For example, We looked for a Wink hub device, we were able to find the FCCID number of the device and searched for it. We found its detailed internal image here. It was found that the device contains an SPI Flash, as shown in the image below.

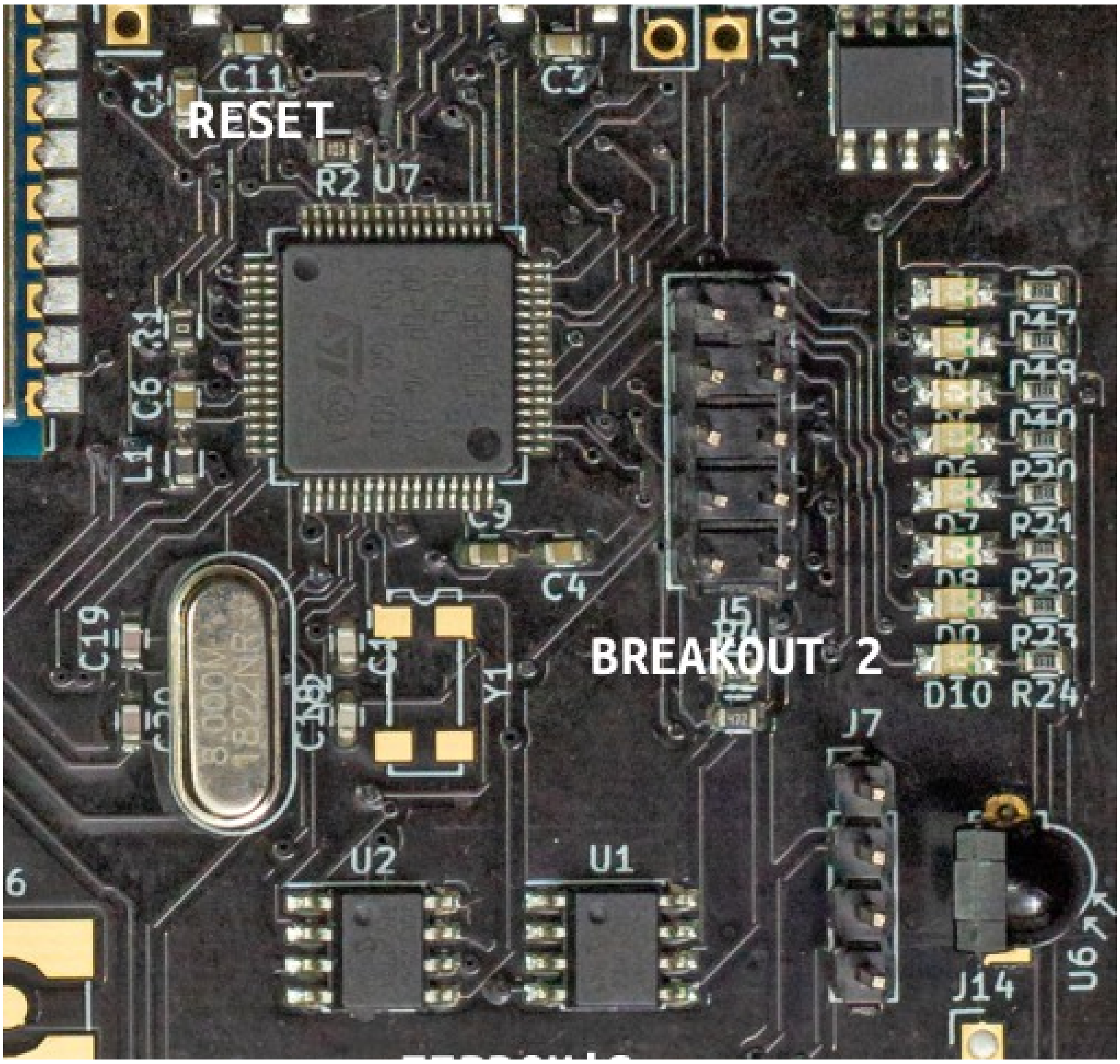

SPI Flash SSP3
FLEXTRONICS
SYS_RESET
12Vdc reg


SPI Flash SSP3

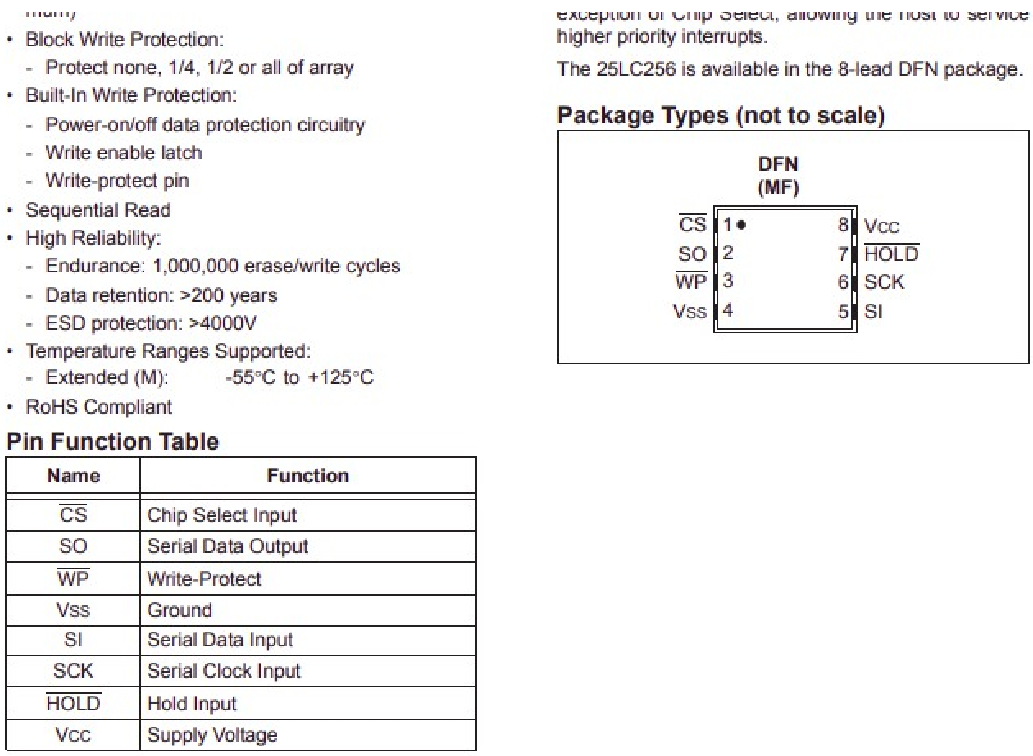
Voila!! We found an SPI chip on the device. Let us go, fetch & attack :)

Case 2: You got access to the hardware.
Once the hardware of a device is found, the first step should be to perform reconnaissance. Inspect each chip, test points present on the printed circuit board (PCB).

- If successful in identifying the SPI chip used on the device, search for its datasheet on the Internet, identify the pin diagrams given, and trace those pins on the PCB. Check the continuity between the controller's SPI pins and the test points present on the PCB using Multimeter. For example, We are showing here the image of the EXPLIoT DIVA board.


RESET
C11
C1
R2
U7
C3
J10
U4
R1
C6
L1
C9
C4
C19
C20
C18
Y1
J5
BREAKOUT 2
D10
R24
R17
R18
R19
R20
R21
R22
R23
J7
U2
U1
J14
U6
6

It contains some chips on the PCB. We googled the datasheet for each chip and found that one of its chips denoted as U2 is an **SPI EEPROM 25LC256**.After finding the chip, the next task is to identify that if the SPI pins (MISO, MOSI, SCK, CS) of the chip is connected to some points on the PCB or not.
Fetching the datasheet of 25LC256, we identified the respective pins of the chip as shown in the image below (Source - 25LC256 datasheet)



mum)

- Block Write Protection:
  - Protect none, 1/4, 1/2 or all of array
- Built-In Write Protection:
  - Power-on/off data protection circuitry
  - Write enable latch
  - Write-protect pin
- Sequential Read
- High Reliability:
  - Endurance: 1,000,000 erase/write cycles
  - Data retention: >200 years
  - ESD protection: >4000V
- Temperature Ranges Supported:
  - Extended (M): -55°C to +125°C
- RoHS Compliant

Pin Function Table

Name	Function
$\overline{CS}$	Chip Select Input
SO	Serial Data Output
$\overline{WP}$	Write-Protect
Vss	Ground
SI	Serial Data Input
SCK	Serial Clock Input
$\overline{HOLD}$	Hold Input
Vcc	Supply Voltage

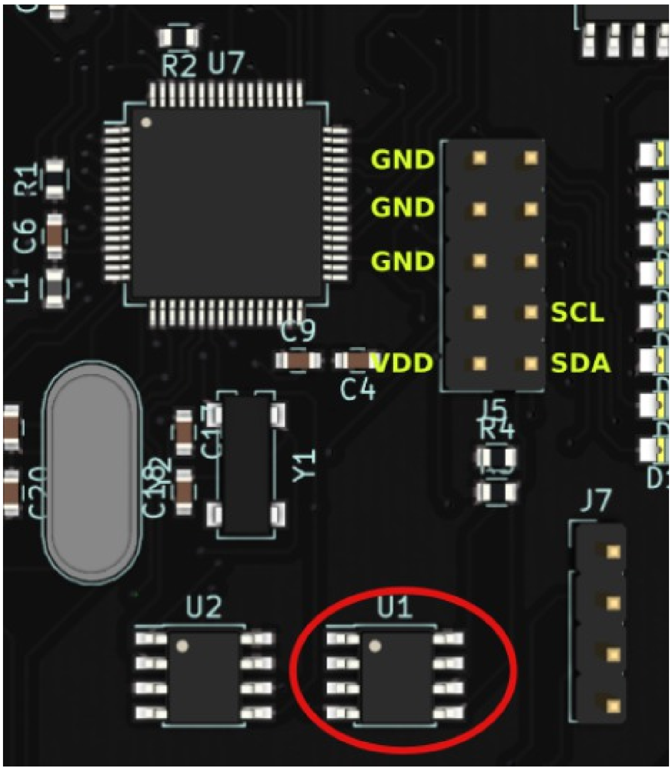
exception of Chip Select, allowing the host to service higher priority interrupts.

The 25LC256 is available in the 8-lead DFN package.

Package Types (not to scale)

DFN
(MF)
$\overline{CS}$ 1
SO 2
$\overline{WP}$ 3
Vss 4
8 Vcc
7 $\overline{HOLD}$
6 SCK
5 SI

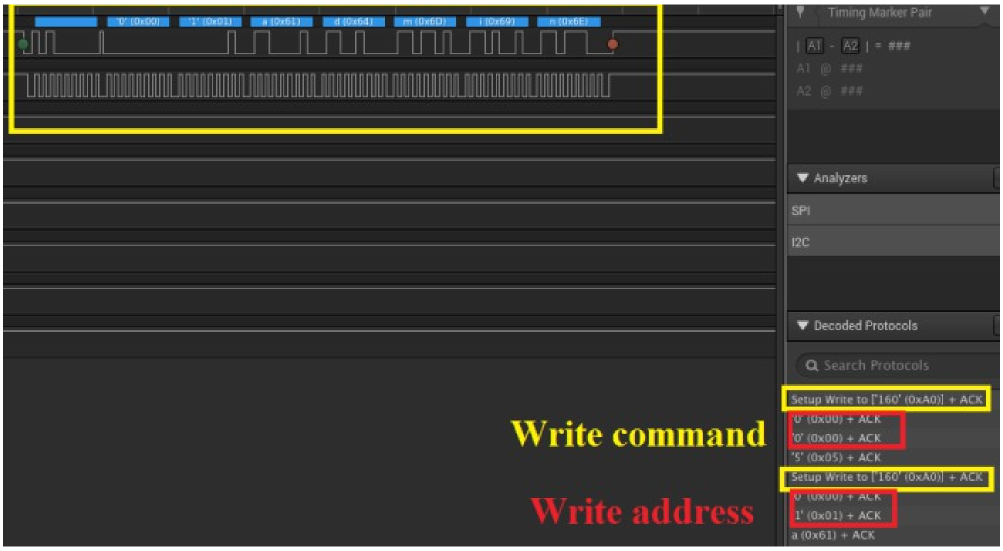
Using the multimeter, we checked the connectivity of these SPI pins with other pins on the PCB. We found that pins on the J5 header, as in the image, below were connected to these SPI pins.


R2
U7
R1
C6
L1
C9
C4
GND
GND
GND
SCL
VDD
SDA
J5
R4
Y1
C20
J7
U2
U1

Yayy!! We can now use these pins to sniff, extract and attack the device :)

3.2 Sniffing SPI Communication

Now, once the SPI pins on the hardware are successfully identified, we can go ahead and sniff the communication between the SPI chip and the controller.
Tools like [Logic Analyzer] can be used to sniff the communication on the SPI bus.



```
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
'0' (0x00) + ACK
'5' (0x05) + ACK
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
'1' (0x01) + ACK
a (0x61) + ACK
```

The logic analyzer shows the signals on all the four SPI pins i.e., MISO. MOSI, CLK, CS. Decoding these signals as per the protocol timing diagram. Softwares like Saleae Logic Analyzer, PulseView have the feature to detect these protocols that directly shows you the decoded data w.r.t the signals.
Yayy!! If the data transfer is in plain text, we can sniff and get it.

3.3 Extracting data/firmware from SPI Memory Chip

To communicate with the SPI chip present on the hardware, we need protocol adapter tools that support SPI protocol and framework to make it feasible to communicate those chips without the host machine. (Sometimes, we also need to desolder the flash memory chips from the hardware, solder it on perf board or put it in a suitable TSOP socket and connect the requited pin to the adapter.)

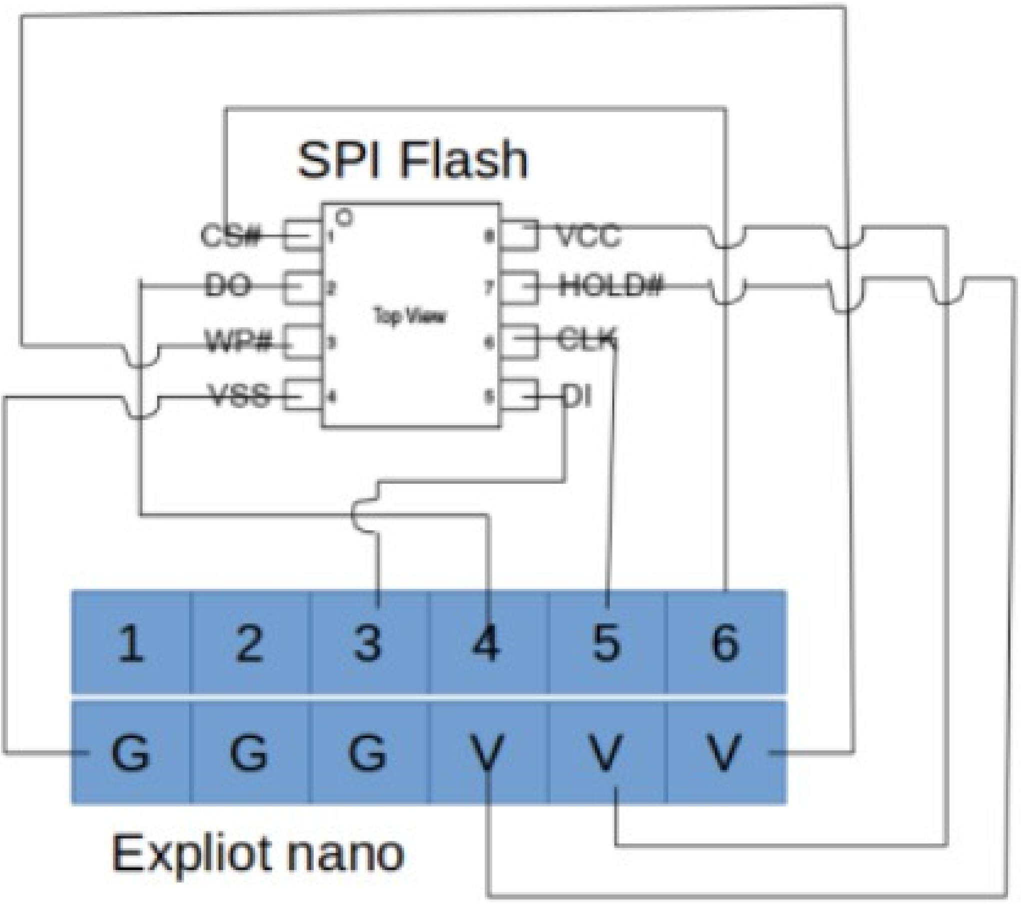
There are various tools and frameworks available to extract the data from the memory chips. A few of them are mentioned below.
1. EXPLIoT Nano
2. Bus Pirate
3. Shikra
4. CH341A
5. EXPLIoT Framework
6. Flashrom
7. pyspiflash
8. pyftdi
9. RaspberryPi (refer here) or Beaglebone (refer here) can also be used.

Depending on the kind of memory chip that we have, like EEPROM or FLASH, and corresponding support by any of these devices, we can select a set of tools to extract the date/firmware. First, we need to make sure whether the SPI chip that we have is supported by these tools. If it is supported and voltage level matches, we proceed with that tool or framework. In case the voltage level of the adapter tool and the SPI chip do not match, use level shifters.

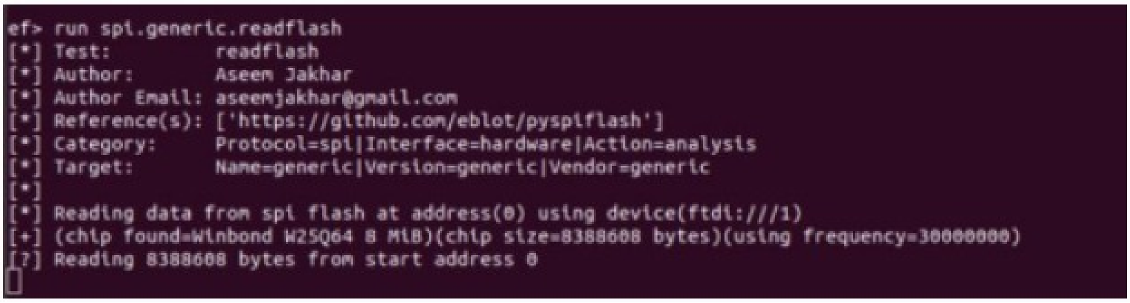
Case 1: You are successful in finding the tools or framework that can support your SPI memory chip

Let us say we find an SPI Flash memory in a camera's hardware, and we want to extract the firmware from the chip. Our approach would be as mentioned below.

1. Take out the datasheet of the SPI flash memory. Identify the pins and voltage required.
2. Accordingly, choose the adapter tool that can be used. Connect the pins of the SPI flas hmemory to the suitable adapter, for example, EXPLIoT Nano as per its datasheet/pinout manual. The connection image is shown below.



1. Then, in your host machine, install the framework supported by the adapter that has been used. For example, EXPLIoT Nano works with EXPLIoT Framework; we set up this framework and use its plugin `run spi.generic.readflash -w camera_dump.bin` to extract the firmware from the flash memory. An example image is shown below.



```
ef> run spi.generic.readflash
[*] Test:          readflash
[*] Author:        Aseem Jakhar
[*] Author Email:  aseemjakhar@gmail.com
[*] Reference(s): ['https://github.com/eblot/pyspiflash']
[*] Category:      Protocol=spi|Interface=hardware|Action=analysis
[*] Target:        Name=generic|Version=generic|Vendor=generic
[*]
[*] Reading data from spi flash at address(0) using device(ftdi:///1)
[+] (chip found=Winbond W25Q64 8 MiB)(chip size=8388608 bytes)(using frequency=30000000)
[?] Reading 8388608 bytes from start address 0
```

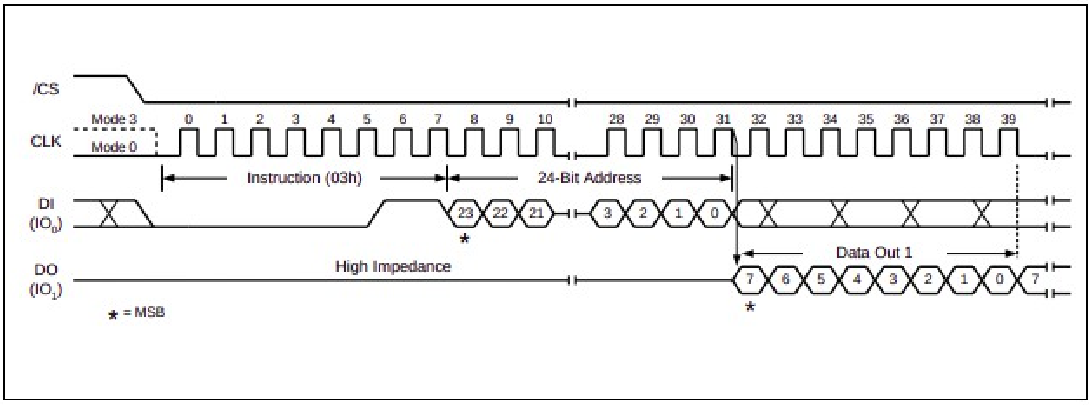
Similarly, other tools and frameworks as suited can be used to dump the data/firmware from the SPI memory chip.

Case 2: You cannot find any tool/framework that can support your SPI memory chip
In this case, the hardware setup steps would be the same as in the above. Take any FTDI232H based adapter, match the voltage levels of the SPI chip and the adapter. Do connection as per adapter and the chip's user manual/datasheet.

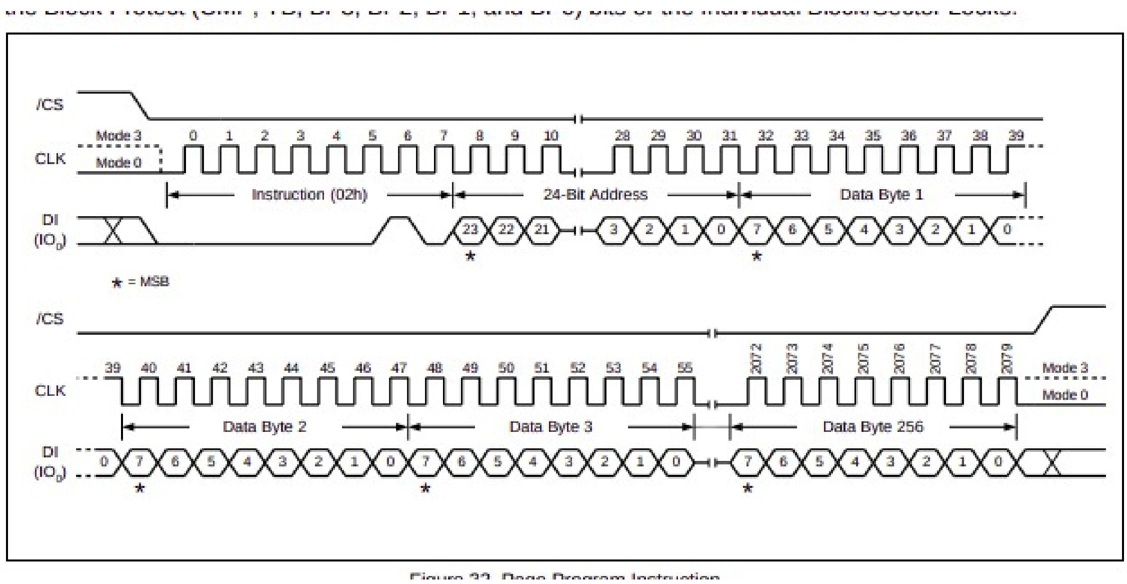
In this case, you need to write your framework, or you can even add the currently unsupported flash chips to any of the open-source frameworks of your choice.

So, for this, you first need to read the datasheet of the SPI memory chip carefully. The datasheet contains all the details required to write your script to read/write the data from/to the memory chip. For example, SPI Flash memory W25Q256JV, let us have a glance at its datasheet. It gives the detailed instructions and commands required to read, write, and perform other actions.
The image below from W25Q256JV datasheet, shows the read data instruction diagram.

The Read Data (03h) instruction is only supported in Standard SPI mode.



Similarly, the below image shows SPI write instruction.



So, following the SPI memory chip datasheet instructions, one can write a script to dump or program the chips. It can also be done using Raspberry Pi or beaglebone. We will talk about detailed SPI memory extraction via real-world example scenarios, different kinds of issues we face while dealing with specific SPI chips, and how to solve those issues in upcoming chapters.

4. Measures that can mitigate the attacks

- Encrypt data on the memory chip to prevent leakage of sensitive info.
- Do not hardcode sensitive data on the memory chips.
- Store encrypted firmware/data
- Physical hardening and protection of chips
  There are also a few crypto-based memory chips like ATAES132A and some other chips as given on Microchip website. Their datasheets claim to provide authentication and encryption features; secure read/write access. Using crypto-based memory chips can increase the complexity of the attacker to steal the data.

5. Measures that can mitigate the attacks

In this chapter, we learned about the SPI protocol, its application, the possibilities of attack scenarios, the methods to attack, and few preventive measures. We hope you enjoyed and got some valuable information out of it. These tools and attack methods would give you a basic understanding of what to do when finding an SPI chip on the device hardware. With that, we hope, pretty soon, you get to say, “Voila!! I found an attack surface in the hardware, let’s attack”.

I2C
Chapter 16: Hardware Attack Surface - I2C

In the previous chapter, we had discussed the SPI protocol. This chapter will discuss another hardware communication protocol, Inter-Integrated Circuit (I2C). So, if you are a beginner in hardware hacking and want to get started with basics, you are at the right place.

1. Introduction

I2C is a synchronous serial communication interface. It is primarily used for short-distance intraboard communication. It was developed by Philips Semiconductor(now NXP Semiconductors). It provides a half-duplex communication mode (i.e., sender and receiver can transmit/receive the data one at a time, not simultaneously). It has a master-slave architecture that supports multiple masters and multiple slaves. It is a 2-wire serial interface.

It consists of two bidirectional open-collector(i.e., the device can pull the respective lines low but cannot drive it high. Hence, this avoids any contention between the devices when one is trying to drive the line high, and the other is trying to pull it low.), lines pulled up with resistors (it keeps the line high when the line is not pulled low by any device):

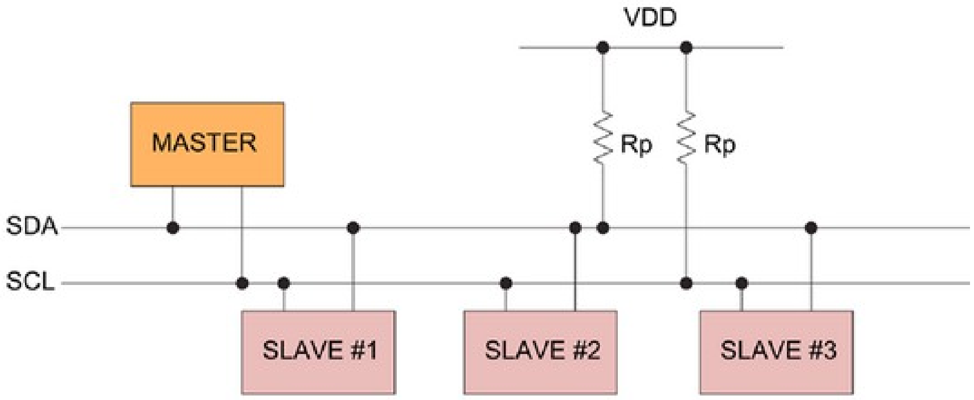
- Serial Clock (SCL)
- Serial Data (SDA)

It supports different bidirectional speed modes 100 kbit/s standard mode, 400 kbit/s fast mode, 1 Mbit/s fast mode plus, 3.4 Mbit/s high-speed mode, and unidirectional 5 Mbit/s ultra fast-mode (these bit rates are without any clock stretching or any other overhead). Multiple masters can exist simultaneously on the same bus. The I2C bus has two nodes:

- Master node -
- This node generates the clock and starts the communication with slaves.
- Slave node -
- It receives the clock and communicates with the master if addressed by it.

Each device connected to the I2C bus has a unique address. The number of nodes on the bus is determined by the address space and the total capacitance of the bus, i.e., 400 pF (Source - wikipedia). It has a 7-bit or 10-bit address space. Further on, we will be discussing w.r.t 7-bit address space. In 7-bit, the maximum number of devices that can be connected over I2C bus is 128. Apart from data bits, I2C has START and STOP bits that indicate the beginning and end of the communication between the master and the slave.

The image below shows the general connection diagram in I2C (Source - here).



1. Master initiates the communication by sending the START bit (SDA line is pulled low when SCL is high, then the SCL is pulled low) followed by the 7-bit/10-bit address of the slave it wants to communicate with, it is followed by 1-bit indicating if it wants to write to (0) or read from (1) the slave.
2. If the slave exists, it responds to the master with an acknowledgment bit ACK.
3. Then, depending on the read or write operation, the master continues to either transmit or receive, and the slave operates in a complementary way, i.e. receive or the transmit, respectively. The size of the data frame that is transmitted is 8-bit. The most significant bit (MSB) of the data byte is transferred first. While transferring, the data is put on the SDA line after SCL is pulled low. During the high period of SCLK, data on the SDA line should be stable. The HIGH or LOW state the SDA line changes only when the SCL line is LOW.
   - In the case of the write operation, the master repeatedly transmits the data, and the slave sends the ACK bit after every received data byte.
   - In the case of the read operation, the master repeatedly receives the slave's data and sends the ACK bit after each received byte except the last one.
   - This transaction can have multiple messages. In the case of the combined format of I2C protocol where the master has to perform multiple reads or write to one or more slaves, instead of sending a STOP bit, the master sends another START bit to retain the bus's control for another message. It is called repeated START bits.
4. If the transaction has to be ended, the master terminates it by sending a STOP bit (SDA line is set high after the SCL is set high).

The image below shows bit transfer on the I2C bus (Source - Standard Doc).

FIGURE 6-1: BYTE WRITE

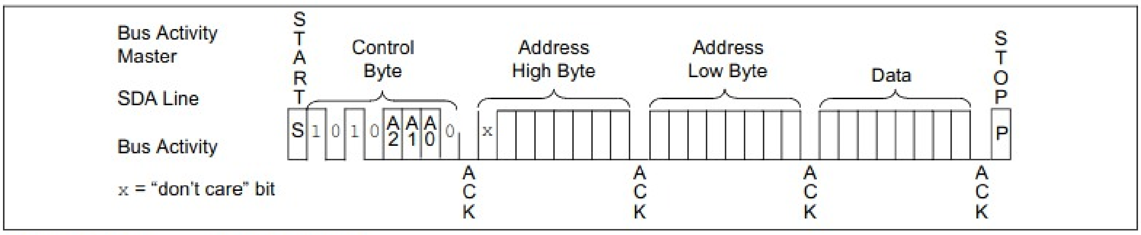
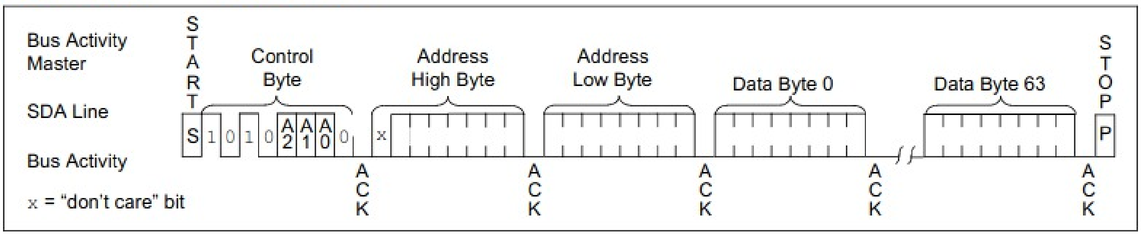
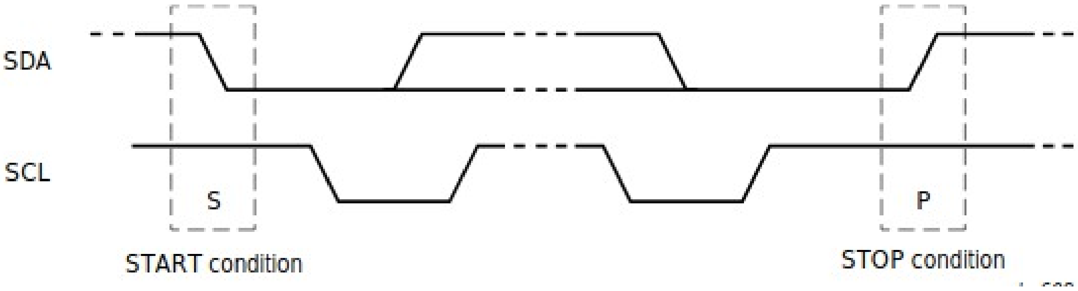

Bus Activity Master
SDA Line
Bus Activity
x = "don't care" bit
START
Control Byte
Address High Byte
Address Low Byte
Data
STOP
S 1 0 1 0 A2 A1 A0 0
x
P
ACK
ACK
ACK
ACK

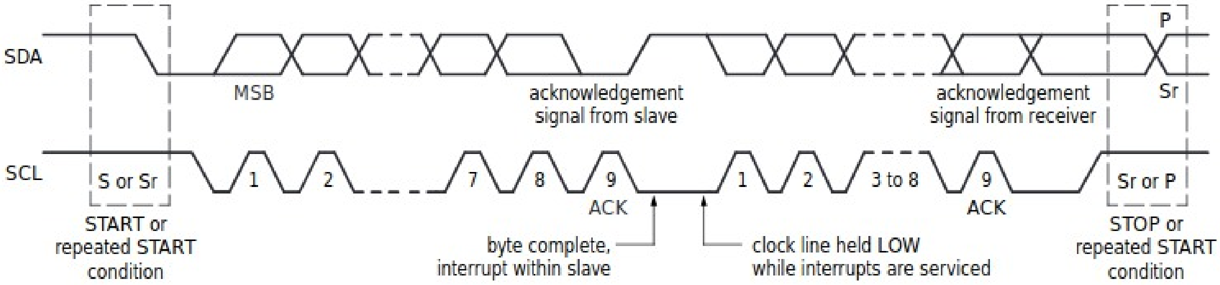
FIGURE 6-2: PAGE WRITE


Bus Activity Master
SDA Line
Bus Activity
x = "don't care" bit
START
Control Byte
Address High Byte
Address Low Byte
Data Byte 0
Data Byte 63
STOP
S 1 0 1 0 A2 A1 A0 0
x
P
ACK
ACK
ACK
ACK
ACK

The below image shows the START and STOP conditions on the I2C bus (Source - Standard Doc).


SDA
SCL
S
P
START condition
STOP condition

The below image shows the data transfer on the I2C bus (Source - Standard Doc).


SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
P
Sr
S or Sr
1
2
7
8
9
ACK
1
2
3 to 8
9
ACK
Sr or P
START or repeated START condition
byte complete, interrupt within slave
clock line held LOW while interrupts are serviced
STOP or repeated START condition

If the slave cannot transmit or receive another data byte due to the slave being involved in some other process like servicing some internal interrupt, it can hold the SCL line low to keep the master in a wait state. Data transfer is continued when the slave is ready to receive/transmit another byte, and it releases the SCL line.

The receiver sends the acknowledgment bit on every received byte. **ACK bit** (i.e., receiver pulls the SDA line low in the 9th clock cycle after the transfer of 8-bit data) acknowledges the transmitter that the data has been successfully received, and the following data can be transferred.
In the given five conditions when:

1. there is no receiver present on the bus with the transmitted address
2. receiver is busy performing some real-time functions
3. receiver receives such data or commands that it does not understand
4. receiver cannot receive any more data bytes
5. the master-receiver has to send the end of the transfer signal to the receiver

In these cases, the transmitter receives no acknowledgment signal **NACK** (i.e., the SDA line remains high during the 9th clock pulse). The master can then either send the STOP signal to end the transfer or send a repeated START signal to start a new transfer.

I2C protocol includes collision detection and arbitration to prevent data corruption in the case; multiple masters simultaneously start the data transfer. **Clock synchronization and arbitration** are used to decide which master will take control of the bus and complete the transmission when multiple masters begin transmitting on the same bus simultaneously. In the case of a single master system, it's not required. The detail working about the Clock synchronization and arbitration can be read from the standard document.

Another essential feature of the I2C protocol is **Clock stretching** There are cases when a slave device to which the master is communicating is not ready to perform the next transaction with the master. This can be when the previous operation on the slave side has not been completed yet. In this case, the slave device can perform clock stretching, i.e., the slave can hold the SCL line low after reception and acknowledgment of byte that makes the master wait until the slave releases the SCL line to high. I2C protocol has many applications. They are used to communicate controllers with sensors, actuators, memory chips, displays, real-time clocks, digital tuning, signal processing circuits, etc. If an adversary gets access to the device's hardware part, there are several ways in which the I2C bus can be exploited if not implemented properly. In further sections of this chapter, we will discuss the attacking cases and methods. The attacking methods would be very similar to what we discussed in the previous chapter of this e-book.

2. Possible Attack Scenarios

- Sniffing the communication between an I2C device like sensors, controlling device, memory chip, ADC, DAC, etc., and the controller/processor. It may lead to the stealing/leak of sensitive information by the attacker. As also discussed in the previous chapter, for example, in some of the embedded devices, an I2C EEPROM is used to store some sensitive information, keys, logs, etc. This sensitive info can be leaked if an attacker sniffs the communication between the EEPROM and the controller/processor. After getting this sensitive info, an attacker can damage in many ways and may be on a large scale if the vendor of the hardware has used the same sensitive info like the same keys in all the hardware in the production.
- Patching the data in the I2C based memory chips, sensors, or any controlled device. In this case, if the data is not securely managed on the device, an attacker can manipulate it. It may result in a malfunction of the device. Depending on the application, manipulation of sensor data can cause significant damage to the system.
- There are also clock glitching related attacks like modifying the frequency of the I2C clock signal. Refer to this research paper to get a more in-depth insight into the attacks related to I2C.

3. Attacking hardware via I2C

There are various ways in which the attacker can attack the hardware via I2C. Few methods to perform the attacks are explained in the below section. It is similar to that explained in the previous chapter.

3.1 Recon

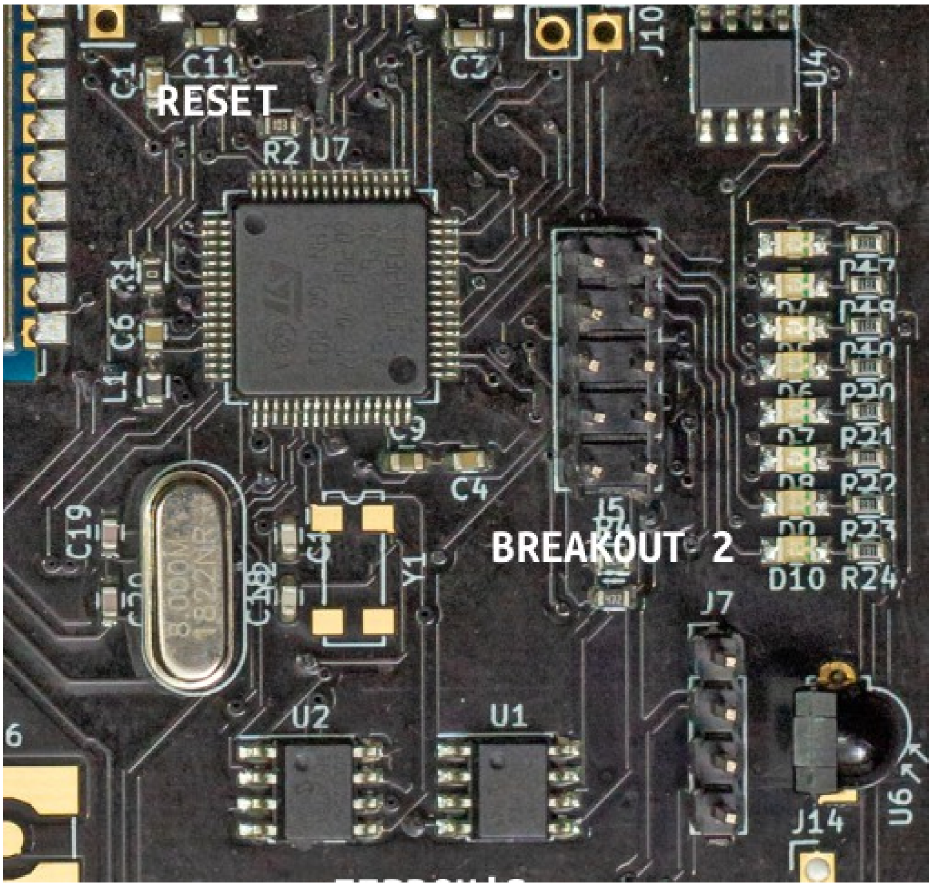
Case 1: You do not have the actual hardware, but you know the FCCID number of the device.

Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. Seeing the internal image, you may get the hint for the presence of an I2C chip or any I2C based device on the hardware. Yayy!! Once you know that you have the attack surface, you can get/purchase that device and perform further required steps to attack the device.

Case 2: You got access to the hardware.

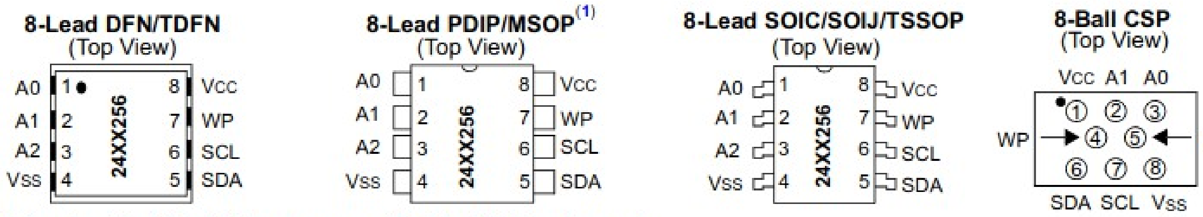
Once the device's hardware is found, the first step should be to tear down the device and perform reconnaissance. Inspect each chip, test points present on the printed circuit board (PCB) to look if you can get access to the I2C bus.

- If successful in identifying the I2C chip used on the device, search for its datasheet on the Internet, identify the pin diagrams given, and trace those pins on the PCB. Check the continuity between the I2C pins of the controller and the test points present on the PCB using Multimeter. For example, We are showing below the image of EXPLIoT DIVA board that we had also used in our previous chapters.


RESET
C11
C1
C3
J10
U4
R2 U7
R1
C6
L1
C9
C4
C19
C20
C18
C17
Y1
BREAKOUT 2
J7
D10 R24
U2
U1
J14
U6

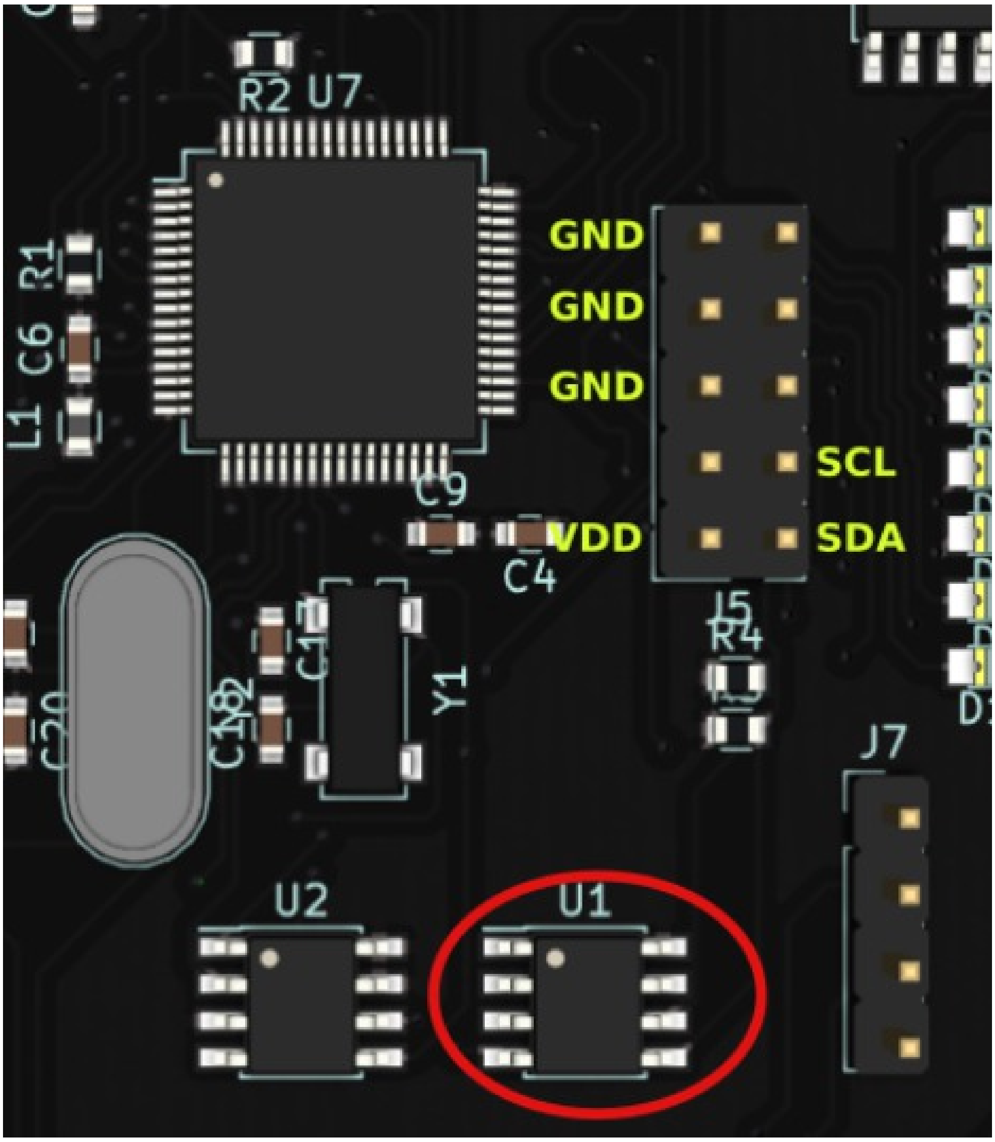
It contains some chips on the PCB. We googled the datasheet for each chip and found that one of its chips denoted as U1 is an **I2C EEPROM 24LC256.**
After finding the chip, the next task is to identify if the I2C pins (SDA, SCL) of the chip is connected to some points on the PCB or not.

Fetching the datasheet of 24LC256, we identified the respective pins of the chip as shown in the image below (Source - 25LC256 datasheet)


8-Lead DFN/TDFN
(Top View)
A0 1 8 Vcc
A1 2 7 WP
A2 3 6 SCL
Vss 4 5 SDA
24XX256
8-Lead PDIP/MSOP(1)
(Top View)
A0 1 8 Vcc
A1 2 7 WP
A2 3 6 SCL
Vss 4 5 SDA
24XX256
8-Lead SOIC/SOIJ/TSSOP
(Top View)
A0 1 8 Vcc
A1 2 7 WP
A2 3 6 SCL
Vss 4 5 SDA
24XX256
8-Ball CSP
(Top View)
Vcc A1 A0
WP
① ② ③
④ ⑤
⑥ ⑦ ⑧
A2
SDA SCL Vss

Note 1: Pins A0 and A1 are no connects for the MSOP package only.

Using the multimeter, we checked the connectivity of these I2C pins with other pins on the PCB. We found that pins on the J5 header, as in the image below, were connected to these I2C pins.

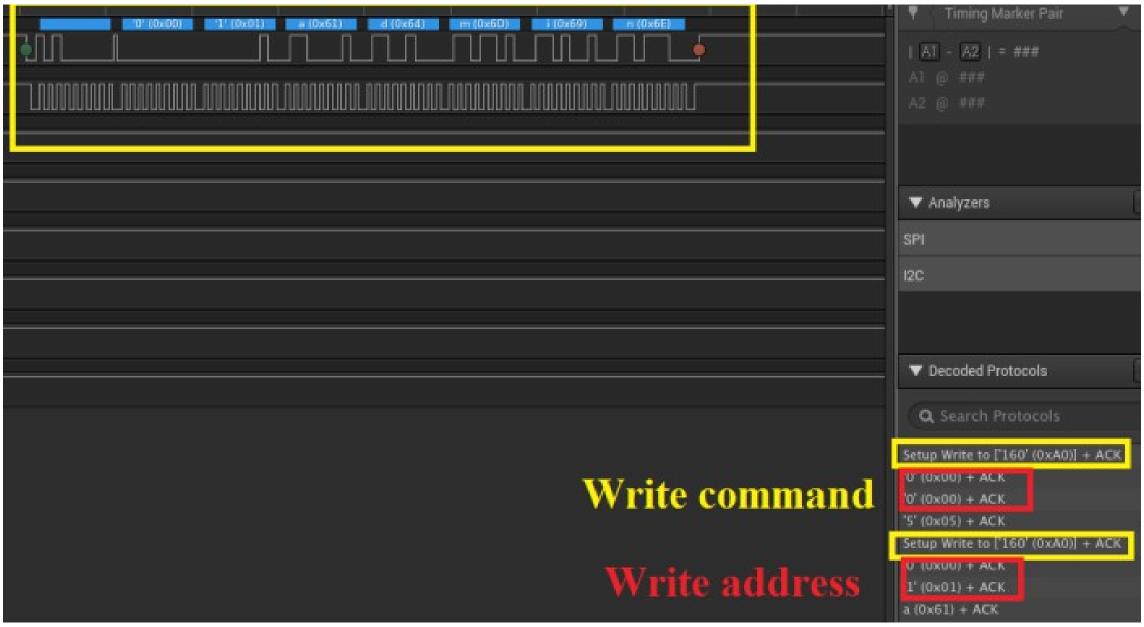

R2 U7
L1 C6 R1
GND
GND
GND
SCL
VDD
SDA
C9
C4
J5
R4
C17
C18
C20
Y1
D1
J7
U2
U1

We have a goodie for you :) , EXPLIoT has Bus Auditor. It helps scanning and identifying debugging and communication interfaces exposed on any hardware board. Its demo example in case of I2C can be read in the blog here

Yayy!! We can now use these pins to sniff, extract and attack the device!

3.2 Sniffing I2C Communication

Now, once the I2C pins on the hardware are successfully identified, we can go ahead and sniff the communication between the I2C chip and the controller.
Tools like Logic Analyzer can be used to sniff the communication on the I2C bus.



```
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
'0' (0x00) + ACK
'5' (0x05) + ACK
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
'1' (0x01) + ACK
a (0x61) + ACK
```

The logic analyzer shows the signals on the SDA and SCL lines of I2C. Decode these signals as per the protocol timing diagram. Softwares like Saleae Logic Analyzer, PulseView have the feature to detect these protocols that directly shows you the decoded data w.r.t the signals.

3.3 Extracting data from I2C memory Chip.

To communicate with the I2C chip present on the hardware, we need protocol adapter tools that support I2C protocol, and a framework that can make it feasible to communicate those chips without the host machine. (Sometimes, we also need to desolder the memory chips from the hardware, solder it on perf board or put it in a suitable TSOP socket and connect the required pin to the adapter.)

Similar to as discussed in the [previous] chapter, various tools and frameworks are available to extract the data from the memory chips. A few of them are mentioned below.

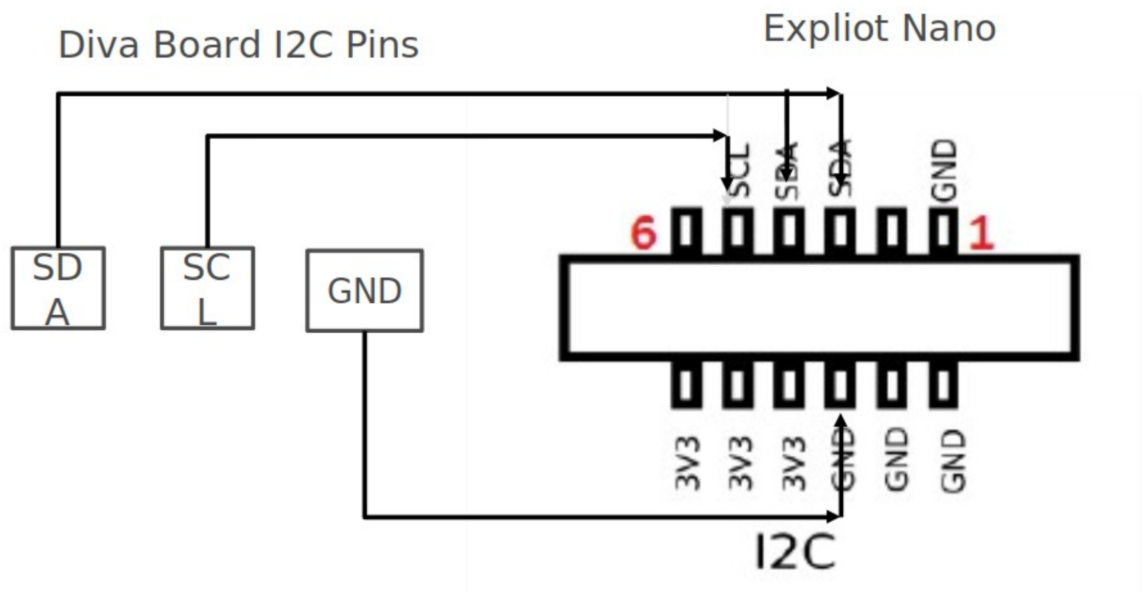
1. EXPLIoT Nano
2. Bus Pirate
3. Shikra
4. CH341A
5. EXPLIoT Framework
6. pyi2cflash
7. RaspberryPi (refer here) or Beaglebone (refer here) can also be used.

After selecting the tool and the framework that suits your requirement, check the tool's voltage levels and the chip. In case the voltage level of the adapter tool and the I2C chip do not match, use level shifters.

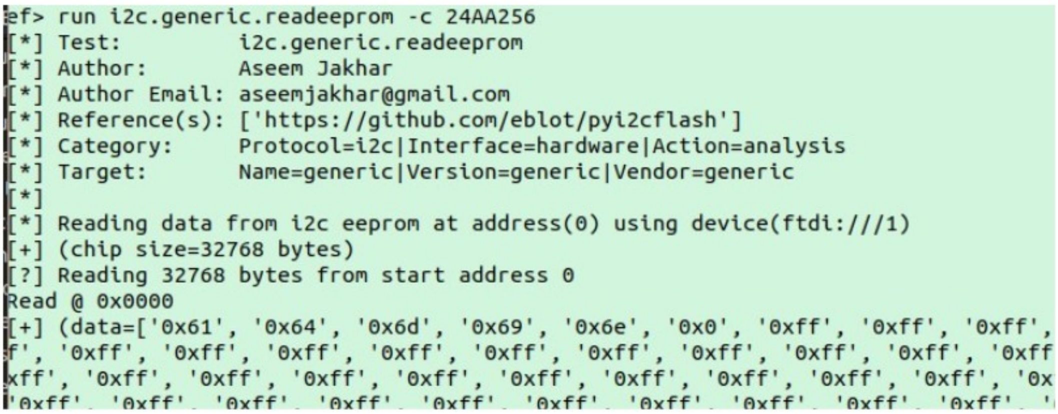
Case 1 You are successful in finding the tools and framework that can support your I2C memory chip

After finding the I2C chip on the device, our approach would be as mentioned below.

1. Take out the datasheet of the I2C chip. Identify the pins and voltage required.
2. Accordingly, choose the adapter tool that can be used. Connect the pins of the chip to the suitable protocol adapter, for example, EXPLIoT Nano as per its datasheet/pinout manual. The connection image is shown below.



3. Then, in your host machine, install the framework supported by the adapter that has been used. For example, EXPLIoT Nano works with EXPLIoT Framework, we set up this framework and use its plugin run `i2c.generic.readeeprom -c <chipname> -w <filename>` to extract the firmware from the flash memory. An example image is shown below.

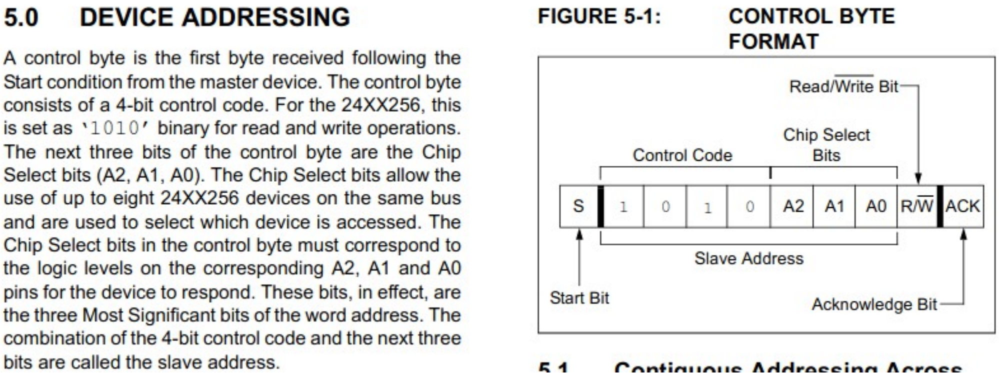


```
ef> run i2c.generic.readeeprom -c 24AA256
[*] Test:          i2c.generic.readeeprom
[*] Author:        Aseem Jakhar
[*] Author Email:  aseemjakhar@gmail.com
[*] Reference(s):  ['https://github.com/eblot/pyi2cflash']
[*] Category:      Protocol=i2c|Interface=hardware|Action=analysis
[*] Target:        Name=generic|Version=generic|Vendor=generic
[*]
[*] Reading data from i2c eeprom at address(0) using device(ftdi:///1)
[+] (chip size=32768 bytes)
[?] Reading 32768 bytes from start address 0
Read @ 0x0000
[+] (data=['0x61', '0x64', '0x6d', '0x69', '0x6e', '0x0', '0xff', '0xff', '0xff',
f', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff
xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0x
'0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '
```

EXPLIoT Framework also supports patching of these I2C memory chips. It has plugin to write on these chips, `run i2c.generic.writeeeprom -c <chipname> -r <patched_filename>` Similarly, other tools and frameworks as suited can be used to dump/patch the data from the I2C memory chip.

Case 2: You cannot find any tool/framework that can support your I2C memory chip
In this case, the hardware setup steps would be the same as in the above case. Take any FTDI232H based adapter, match the voltage levels of the I2C chip and the adapter.
Do connection as per adapter and the chip's user manual/datasheet.

In this case, you need to write your framework, or you can even add the currently unsupported chips to any of the open-source frameworks of your choice. So, for this, you first need to read the datasheet of the I2C memory chip carefully. The datasheet contains all the details required to write your script to read/write the data from/to the memory chip. For example, for I2C memory 24LC256, let us have a glance at its datasheet. It gives the detailed instructions required to read, write, and perform other actions. The image below from 24LC256 datasheet, shows the device addressing method used in this chip.



5.0 DEVICE ADDRESSING

A control byte is the first byte received following the Start condition from the master device. The control byte consists of a 4-bit control code. For the 24XX256, this is set as '1010' binary for read and write operations. The next three bits of the control byte are the Chip Select bits (A2, A1, A0). The Chip Select bits allow the use of up to eight 24XX256 devices on the same bus and are used to select which device is accessed. The Chip Select bits in the control byte must correspond to the logic levels on the corresponding A2, A1 and A0 pins for the device to respond. These bits, in effect, are the three Most Significant bits of the word address. The combination of the 4-bit control code and the next three bits are called the slave address.

FIGURE 5-1: CONTROL BYTE FORMAT

Read/Write Bit
Chip Select Bits
Control Code
S
1
0
1
0
A2
A1
A0
R/W
ACK
Slave Address
Start Bit
Acknowledge Bit

5.1 Contiguous Addressing Across

Similarly, the below image shows I2C write operation in this chip.

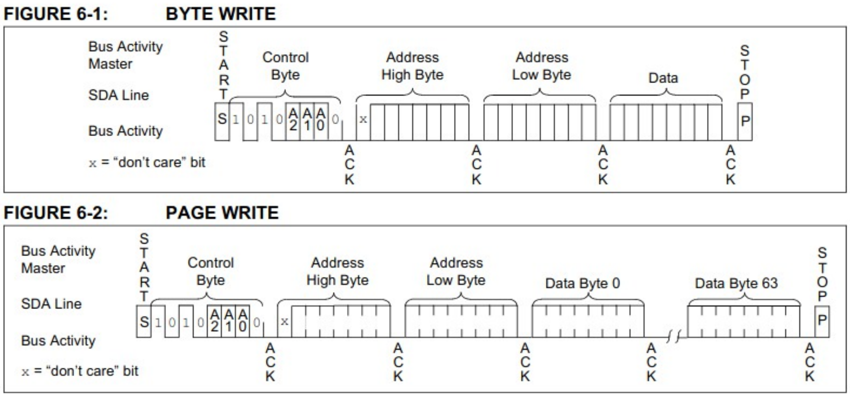


FIGURE 6-1: BYTE WRITE

Bus Activity Master
SDA Line
Bus Activity
x = "don't care" bit
START
Control Byte
Address High Byte
Address Low Byte
Data
STOP
S 1 0 1 0 A2 A1 A0 0
x
P
ACK
ACK
ACK
ACK

FIGURE 6-2: PAGE WRITE

Bus Activity Master
SDA Line
Bus Activity
x = "don't care" bit
START
Control Byte
Address High Byte
Address Low Byte
Data Byte 0
Data Byte 63
STOP
S 1 0 1 0 A2 A1 A0 0
x
P
ACK
ACK
ACK
ACK
ACK

So, by following the I2C memory chip datasheet instructions, one can write a script to dump or program the chips.

4. What measures can make it difficult for an attacker to attack?

- Encrypt data on the memory chip to prevent leakage of sensitive info.
- Do not hardcode sensitive data on the memory chips.
- Store encrypted data
- Physical hardening and protection of chips
- Protection against clock glitching as discussed here
- There are also a few crypto-based memory chips like AT88SC0104CA and some other chips as given on Microchip website. Their datasheets claim to provide authentication and encryption features, and secure read/write access. Using crypto-based memory chips can increase the complexity of the attacker to steal the data.

5. Conclusion

In this chapter, we learned about the I2C protocol, its application, the possibilities of attack scenarios, the methods to attack, and few preventive measures. We hope you enjoyed and got some valuable information out of it. These tools and attack methods would give you a basic understanding of what to do when you find an I2C device on the hardware.

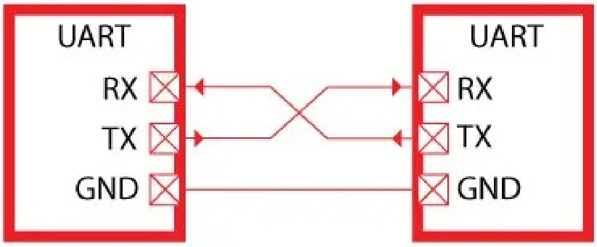
UART
Chapter 17: Hardware Attack Surface - UART

Previously, we discussed UART as a hardware attack surface in the IoT attack surface (Chapter 2 of the e-book). In this chapter, we will discuss it in detail. **Universal Asynchronous Receiver - Transmitter (UART)** is used in most embedded devices. While attacking any device, if you can find a UART port on the hardware, hurray!! You got a way to break into the device. We will cover the introduction to UART, why is it interesting from a hardware attack perspective, how to interact with it after finding a UART port on the device, and the possible attack scenarios.

1. Introduction

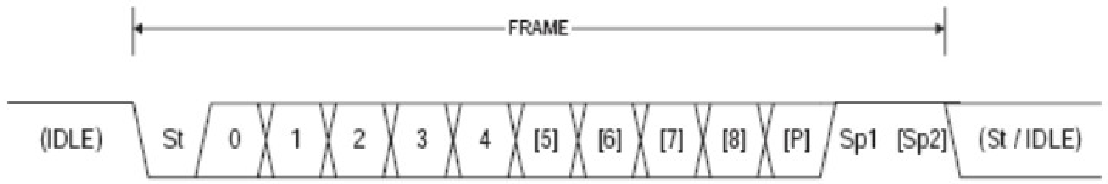
UART interface is a hardware device (physical circuit in the controller or a standalone IC) used for asynchronous serial communication. It enables the translation of data between the serial and parallel interfaces using a shift register. It is the most commonly used in embedded devices. The communication directly takes place between two UARTs. The UART interface on the transmitting side takes in the parallel data and converts it in a serial form. It then transmits the data serially to the UART interface on the receiving side. The receiving UART interface takes in the serial data and converts it back to the parallel to give it to the receiving side's controlling device.

It communicates using two signal lines RX (receiver) and TX (transmitter). The data from the TX pin on the transmitting UART interface is received at the RX pin on the receiving UART interface. The image is shown below (Image Source - here)



During transmission of data packets, START (always logical LOW) and STOP (always logical HIGH) bits are added that defines the start and the end of the data packets. UART communication has one start bit, 5 to 8-bit data (or even 9 bit, if no parity bit used), and 1 stop bit, means 8-bit data transfer once the signal is high to low.

The image for the data frame is shown below (Image Source - here)



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

Each data byte to be transferred has a logic low START bit, data bits, optional parity bit, one or more stop bits. In most cases, the least significant (LSB) of the data bit is sent first. The start bit indicates the receiver regarding the arrival of a new data byte, followed by 5 to 9-bit data. An optional parity bit would be present after all data bits, followed by one or more stop bits that are always logic HIGH. STOP bit indicates the receiver regarding the completion of the data byte sent by the transmitter. More detail can be read from here. The rate at which the data bits are transferred i.e., bits per second, is called baudrate. The transmitting and the receiving UART interface should have the same baudrate, data length, parity bit, and stop bit for proper operation.

UART interface is used in microcontrollers to communicate with the peripheral devices. It is used in various communication devices like modems, Wi-Fi chips, Bluetooth modules, GPS modules, routers, cameras, and many other applications. UART is also used as an alternative debug interface for the embedded devices that do not have interfaces like a keyboard or monitor to debug. In further sections, we look into possible attack scenarios. We will brief you about the tools and a few methods that you can use to attack the device via UART.

2. Possible Attack Scenarios

One of the important aspects of why the UART port is so interesting from the hardware security perspective is how it is used to connect to the debug shell or console of the embedded operating system of the device. If that is not secured properly, it gives the attacker the path to break into the device by getting access to the root shell.

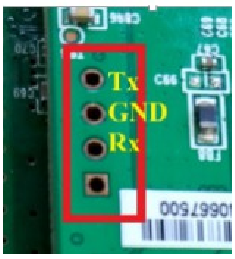
Getting access to the device root shell enables the attacker to (Source - here):

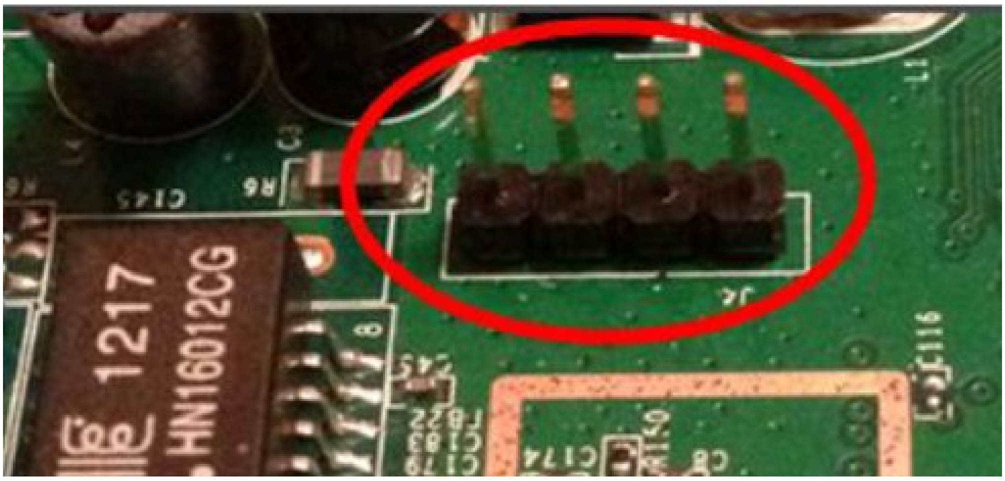
- Hunt around the file system and look for some vulnerable binaries or services running that could be easy to fuzz for an attacker. Sometimes, access to debug or network-related binaries can enable remote access to compromise the device.
- Perform modifications on the running system
- Possibility to flash the patched firmware on the device.
- Explore the file system to find some sensitive and hardcoded values such as encryption keys, configurations, credentials, etc. Getting access to this sensitive info and make it much easier for an attacker to perform further attacks.

Apart from getting access to the root shell, the adversary can sniff the communication between the peripheral device and the controller that is happening via UART serial communication. Also, if device console output is connected to UART, it gives much debugging and system log related info on the console. It can be very useful to the adversary for reverse engineering and knowing about the system's behavior.

3. Attacking hardware via UART interface

To attack the device via a UART port, we first need to identify the UART pins RX, TX on the hardware. Below are a few images showing how most UART port test points/ pinouts on the hardware look.





Source - here

3.1 Recon

Case 1: You do not have the actual hardware, but you know the FCCID number of the device.

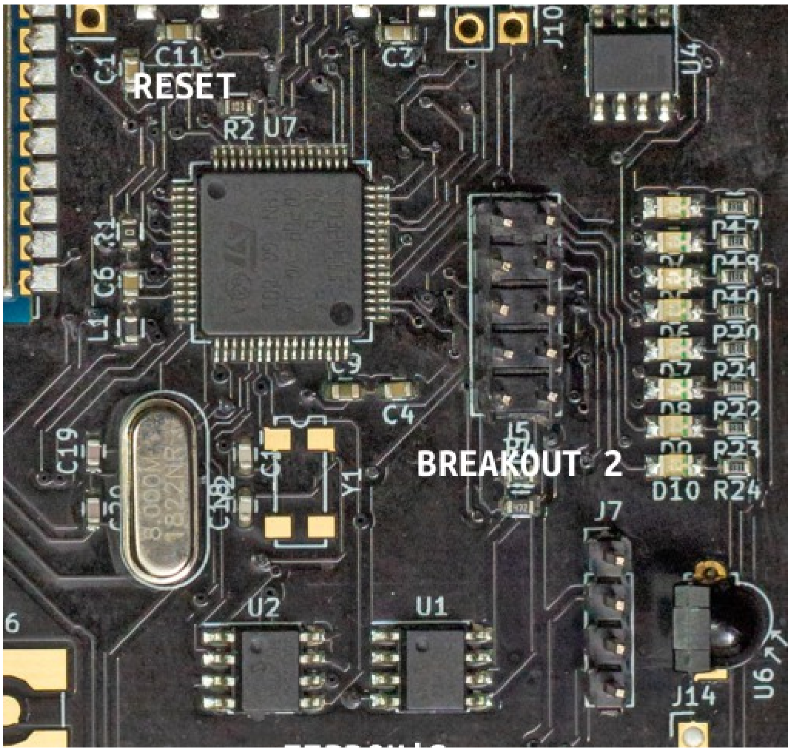
Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. Seeing the internal image, you may get a hint for devices with UART interfaces on the hardware. Yayy!! Once you know that you have the attack surface, you can get/purchase that device and perform further required steps to attack the device.

Case 2: You got access to the hardware.

Once the hardware of a device is found, the first step should be to perform reconnaissance. Inspect each test point, and chips present on the printed circuit board (PCB) to see if you can get a UART interface.

As shown in the images above, UART port usually has 4 pins RX, TX, Vcc, and GND. If we find a set of 4 pins together, we hope it is a UART port. Few ways in which you can manually identify the UART port pins on the device are mentioned below:

1. Digital Multimeter conductivity test - Identify the microcontroller used in the device, take out its datasheet, and identify the microcontroller's UART pins. For example, we will take the example of the EXPLIoT DIVA board.



The microcontroller used in the diva board marked as U7 on the PCB is **STM32F411x**, we can find in its datasheet about the pins having UART interface connection. The image below shows that section of the datasheet.

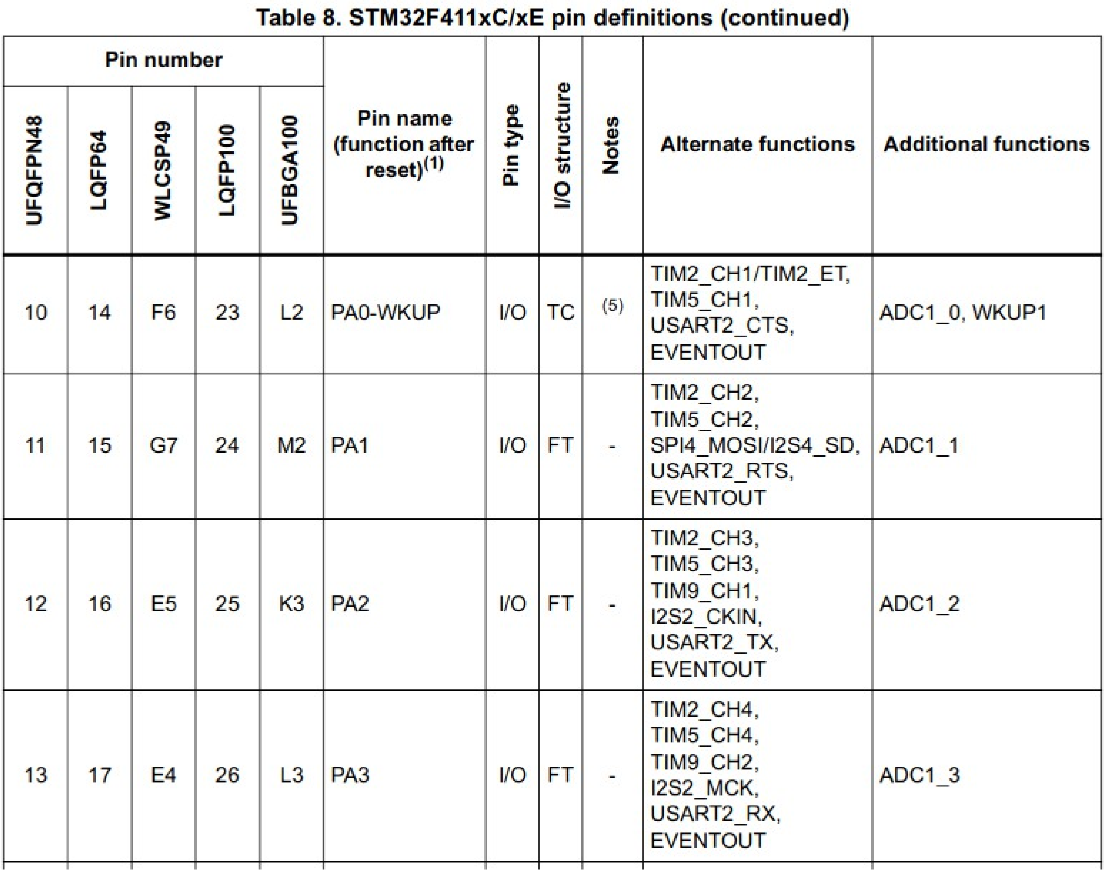
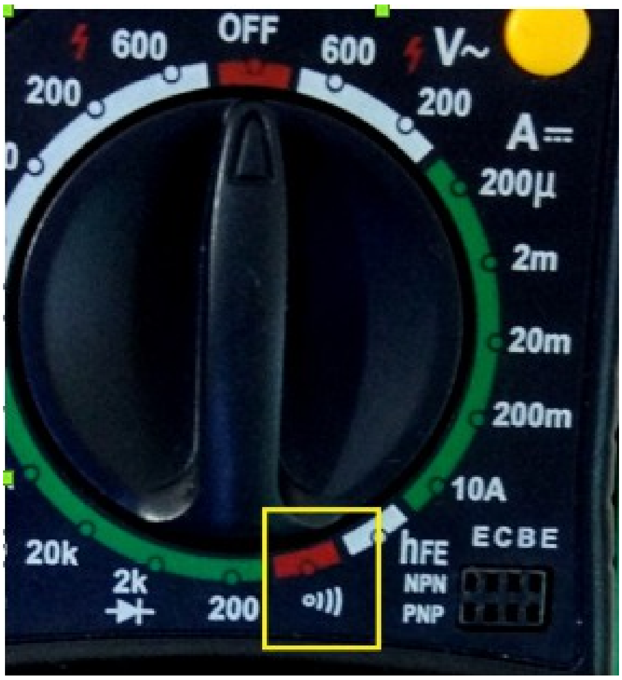


Table 8. STM32F411xC/xE pin definitions (continued)

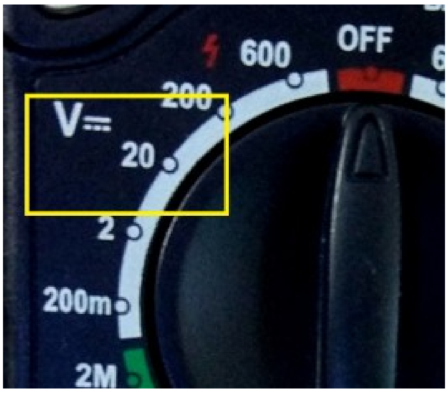
Pin number					Pin name (function after reset)(1)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	LQFP64	WLCSP49	LQFP100	UFBGA100						
10	14	F6	23	L2	PA0-WKUP	I/O	TC	(5)	TIM2_CH1/TIM2_ET, TIM5_CH1, USART2_CTS, EVENTOUT	ADC1_0, WKUP1
11	15	G7	24	M2	PA1	I/O	FT	-	TIM2_CH2, TIM5_CH2, SPI4_MOSI/I2S4_SD, USART2_RTS, EVENTOUT	ADC1_1
12	16	E5	25	K3	PA2	I/O	FT	-	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, USART2_TX, EVENTOUT	ADC1_2
13	17	E4	26	L3	PA3	I/O	FT	-	TIM2_CH4, TIM5_CH4, TIM9_CH2, I2S2_MCK, USART2_RX, EVENTOUT	ADC1_3

Depending on the IC package that is used in the device (here it's LQFP64), we identify the UART port pins on the controller. Then, we set the multimeter in continuity mode as in the image below :



Then, we put one probe on the UART port pins on the controller and the other probe on the test points/ headers pins on the PCB that are suspected to be UART port pins. This test is repeated until pins are identified. Vcc and Ground pins can also be identified similarly.

1. Identify using voltage measurements - In this case, even if we do not have the datasheet of the controller used in the device, we can identify the UART port pins by measuring the voltage on the suspected pins/test points.
   - After opening the device, we look for possible UART port pin/test points groups. Generally, they are four or more pins in the group, as shown in the images above. Mark all such suspected UART port pin groups to test all of them.
   - Starting with identifying the GND pin, set the multimeter in continuity mode. Put the red probe of the multimeter on the suspected ground pin and the other probe on the ground pin of the device's input supply or any metallic shield present on the device. If the multimeter beeps, i.e., if there is continuity between the suspected ground pin and the actual ground pin on the device, it means that it is the ground pin that can be used for UART port connections. If not, then repeat with other pins in the suspected UART port pins group.
   - Vcc is not generally used for connecting to the UART interface but identifying it narrows down our search to find the other two pins Rx and Tx.
     - Power on the device
     - Set the multimeter rotary to DC voltage like V (20) (considering device under 20 v range) as in the image below :



     - Put the black probe on the device's ground pin and the red probe on the suspected Vcc pin. If the voltage reading on the multimeter shows a constant 3.3v or 5v (depending on the device voltage), that's the Vcc pin. If not, then repeat with other pins in the suspected UART port pins group.
   - To identify the Tx pin, power on the device, set the multimeter to dc voltage range as mentioned in the previous point. Put the black probe on the device’s ground pin and the red probe on the suspected pin. The reading on the multimeter shows varying voltage, it means that it is the TX pin. It is because the device keeps transmitting on the TX pins. When the

- Generally, if we identify the three pins Tx, Gnd, Vcc successfully, it becomes easier to identify the fourth pin Rx. While measuring voltage as we did in the previous point, it sometimes shows different results in different cases. Sometimes it shows constant low or high voltage or even varying voltage in some cases.
    1. Automated Scanning - There are tools to perform automated scanning, like Bus Auditor to scan and identifying debugging and communication interfaces exposed on any hardware board. Its demo example, in case of UART interface, can be read in the blog here. Jtagulator, Arduino based UARTFuzz can also be used to scan for UART port pins. Along with scanning, these tools also detect the baud rate.

3.2 UART port interfacing, communication sniffing & device accessing

After identifying the UART port pins on the hardware, now it's time to attack.

To sniff the communication:
Tools like Logic Analyzer can be used to sniff the communication between the device having the UART interface and the controller.

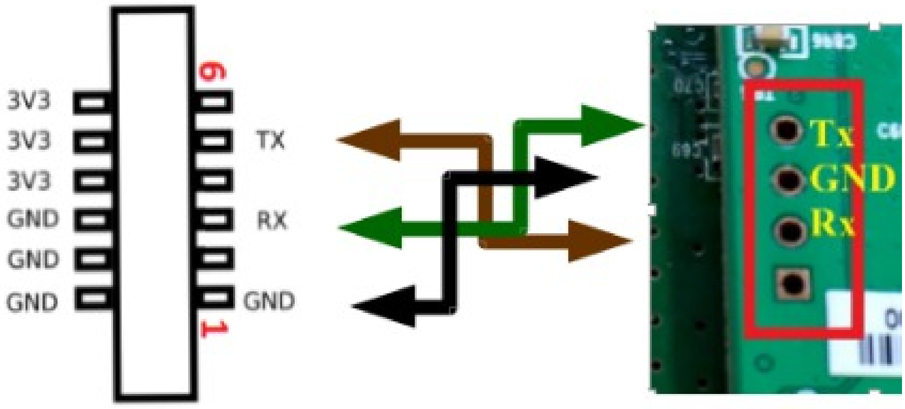
The logic analyzer shows the signals on the RX and TX lines of a UART port. These signals are as per the UART communication protocol structure, as discussed above. Softwares like Saleae Logic Analyzer, PulseView have the feature to detect these UART communication signals that directly shows you the decoded data w.r.t the signals.

3.3 Accessing the device console over UART interface

As discussed above, in most embedded devices, the UART port is used as a debug interface. It is connected to the device’s serial console that can give access to a root shell, log messages. To access the device’s console over UART port, we need an adapter that supports the UART interface. Before using any external tool with the device, we need to match their voltages. Also, while connecting the external tool and the target device, make their grounds common. Few tools that can be used as USB to UART converters for UART interfacing are:

- Expliot nano
- Bus Pirate
- Shikra
- CH341A
- CP2102 USB to TTL covertors

After the tools are selected, do the proper connections as per the datasheet. For example, here we would show using Expliot nano. We will do the connections as per its datasheet /pinout manual. The connection diagram between the device's UART port pins and the Expliot nano is shown below.



The RX of the device is connected to the TX of the adapter, and the TX of the device is connected to the RX of the adapter. The ground of both devices is connected. Now, the device's serial console can be accessed using serial terminals like minicom, picocom, putty, and teraterm. Before connecting, we need to know the baudrate. If the UART port scanning is done using automated tools like bus auditor and jtagulator, we automatically get the baudrate. However, if the UART scanning is done manually using the multimeter or the voltage measurement, we need to find the baudrate. For example, with Expliot nano, we have EXPLIoT Framework that can be used to find UART baudrate using the plugin `run uart.generic.baudscan -p <device_port> -v`. In manual cases, we need to do hit and trials for finding the baudrate.

Once the baudrate is detected, the converter is connected to the host machine via the serial terminals to access the device's serial console. If authentication is not implemented securely in the device, the adversary can directly get into the device after getting access to the root shell. It can be used to attack the device in several ways, as discussed above in the "Possible Attack Scenarios" section.

4. Conclusion

In this chapter, we learned about the UART interface, its application, the possibilities of attack scenarios, and the methods to attack. We hope you enjoyed and got some valuable information out of it. These tools and attack methods would give you a basic understanding of what to do when you find a UART interface on the hardware and play around with it to attack some devices.

JTAG/SWD

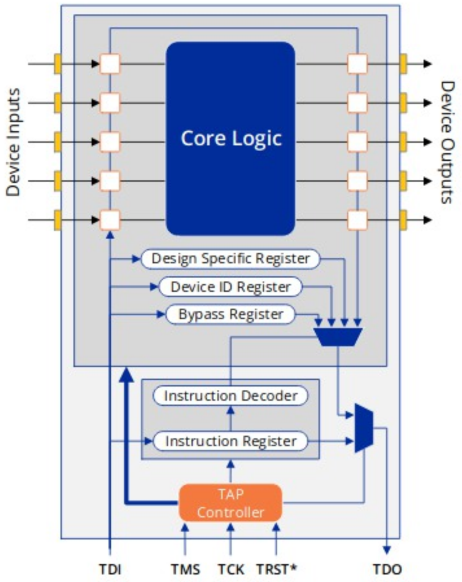
Chapter 18: Hardware Attack Surface- JTAG, SWD

We discussed JTAG and SWD as a hardware attack surface in the IoT attack surface chapter (2nd chapter of this e-book). In this chapter, we will discuss them in detail. Along with their basic introduction, we will discuss the possible attack scenarios, tools, and methods that you can use to attack the device if you get access to the JTAG/SWD interface on the hardware. So, if you are new to it and want to get started, stay tuned!

1. Introduction

1.1 JTAG

JTAG (Joint Test Action Group), an industry-standard, was developed by the association, the Joint (European) Test Access Group, in 1985. It was originally developed for verifying designs and testing of printed circuit boards (PCB). Later in 1990, it was standardized as an IEEE standard 1149.1-1990 as **Standard Test Access Port and Boundary-Scan Architecture.** Initially, with the invention of integrated circuits, increasing complexity, higher density, and smaller components, it was not easy to test the physical interconnections on PCB. JTAG boundary scan came out as a solution to perform the testing and debugging of chips' physical interconnection by limiting physical access to just a few signals. Today, JTAG is used for many other applications, including incircuit debugging, giving access to directly communicate with the memory/registers within the chip without direct external access to the system address or the data bus, and for programming devices. The image below shows the architecture of the JTAG chip (Source - here).

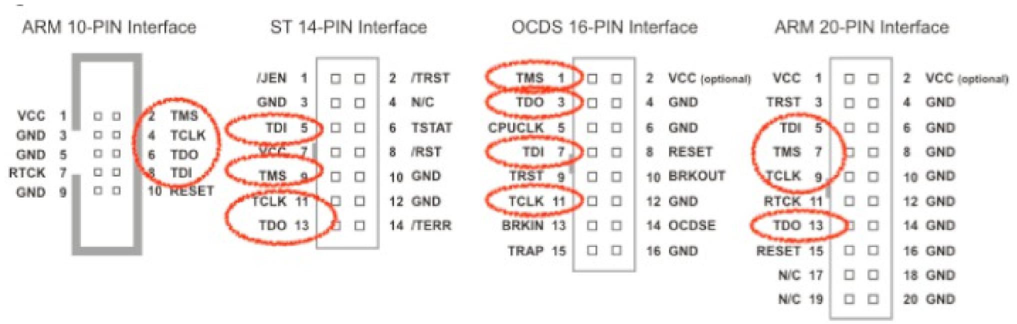

Device Inputs
Core Logic
Device Outputs
Design Specific Register
Device ID Register
Bypass Register
Instruction Decoder
Instruction Register
TAP Controller
TDI
TMS
TCK
TRST*
TDO

The JTAG chip consists of logic cells or boundary-scan cells that connect the chip to the PCB. These can capture data from pin or core logic signals as well as send data onto pins. These logic cells are accessed through a serial test data input (TDI) and test data output (TDO) interface. The test controller's primary interface that provides access to the logic is the Test Access Port (TAP), consisting of 4 required signals and an optional reset signal.

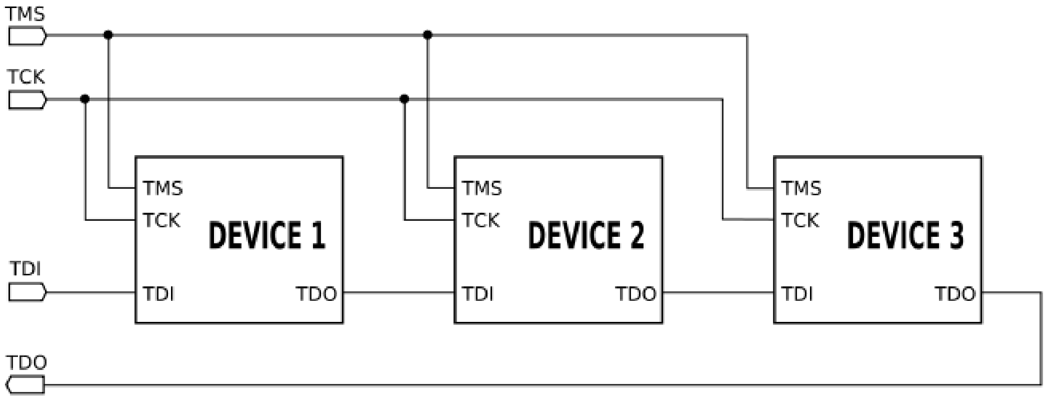
- TCK (Test Clk) - Test clock synchronizes the operations of the internal state machine. The JTAG standard does not specify the actual clock speed.
- TMS (Test Mode Select) - It controls the JTAG state machine. It is sampled at the rising edge of the TCK to determine the next state of the state machine.
- TDI (Test Data In) - Sends data into the chip. When the internal state machine is in the correct state, it is sampled at the TCK's rising edge.
- TDO (Test Data Out) - Data coming out of the chip. When the internal state machine is in the correct state, it is valid on TCK's falling edge.
- TRST (Test Reset : Optional) - Resets the TAP controller state machine.

As defined by the IEEE-1149.1 standard, the TAP controller uses a 16-state finite state machine controlled by TCK and TMS signals. The state of TMS on the rising edge of TCK determines the transition to the next state. Each JTAG TAP has an Instruction Register (IR) and a Data Register (DR). The size of these registers is variable. The state machine selects the operations/ instructions via IR and passes the parameters or data update via DR. The detailed working of the state machine can be read here.

JTAG specification does not have the defined protocol for the connector design. JTAG connectors can be found varying from 6, 10, 14, 16, 20, etc. numbers of pin interfaces. These connectors can have extra signals apart from the 4 JTAG signals. as shown in the image below (Source - Reference)


ARM 10-PIN Interface
VCC 1
GND 3
GND 5
RTCK 7
GND 9
2 TMS
4 TCLK
6 TDO
8 TDI
10 RESET
ST 14-PIN Interface
/JEN 1
GND 3
TDI 5
VCC 7
TMS 9
TCLK 11
TDO 13
2 /TRST
4 N/C
6 TSTAT
8 /RST
10 GND
12 GND
14 /TERR
OCDS 16-PIN Interface
TMS 1
TDO 3
CPUCLK 5
TDI 7
TRST 9
TCLK 11
BRKIN 13
TRAP 15
2 VCC (optional)
4 GND
6 GND
8 RESET
10 BRKOUT
12 GND
14 OCDSE
16 GND
ARM 20-PIN Interface
VCC 1
TRST 3
TDI 5
TMS 7
TCLK 9
RTCK 11
TDO 13
RESET 15
N/C 17
N/C 19
2 VCC (optional)
4 GND
6 GND
8 GND
10 GND
12 GND
14 GND
16 GND
18 GND
20 GND

For communicating with the device via the JTAG interface, we would have to identify the 4 JTAG signals: TCK, TMS, TDI, and TDO, that communicate with the JTAG chip state machine via TAP. It is also possible to communicate with more than one JTAG chips by connecting them in a daisy chain as shown in the image below (Source - Wikipedia)



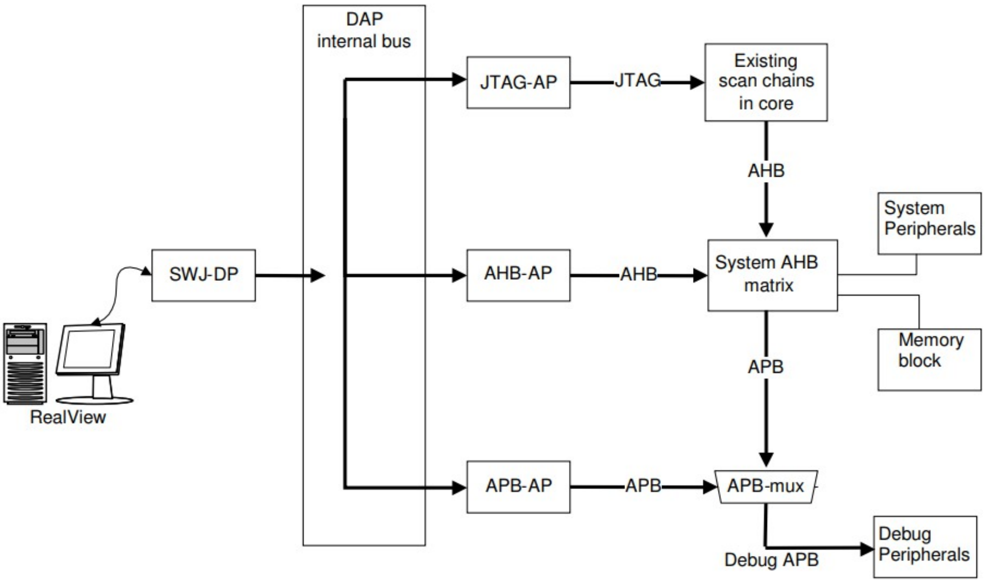
There is a reduced pin count JTAG called compact JTAG (cJTAG) that only has two pins, TMSC (Test Serial Data) and TCKC (Test Clock). It is defined as part of the IEEE 1149.7 standard. JTAG specification defines some mandatory boundary-scan related instructions and some optional instructions. However, the TAPs that are used for debugging applications instead of boundary scan, generally provide minimal or no support for these mandatory instructions related to boundary scan. The two important instructions are:

- BYPASS - TDI and TDO are connected to a single-bit data register (also called BYPASS). This instruction allows the device to get bypassed while allowing the serial data to go through the next devices in the chain/scan path.
- IDCODE - This is an optional instruction but used widely (not universally). It is related to a 32 - bit device ID register (IDCODE). It includes the chip ID i.e., the manufacturer code in the standardized format.

After the reset state, IR is preloaded with BYPASS or IDCODE instruction. Such identification can also help in identifying the kind of processor/microcontroller used in the chip. Other instructions include EXTEST, SAMPLE, PRELOAD, HIGHZ, INTEST, CLAMP, RUNBIST, and USERCODE. The devices provided by the manufacturer may define more instructions. JTAG supports different architectures, including ARM, Atmel AVR, TI MSP430, FPGAs, MIPS, CPLDs, etc.

1.2 SWD

SWD (Serial Wire Debug) provides the debug port by reducing the pin count to just two, the bidirectional data signal (SWDIO), and a clock signal (SWCLK) sent by the host. It provides all the normal JTAG debug and test functionality (it does not provide the boundary scan feature as in JTAG). It uses an Arm standard bi-directional wire protocol, defined in the Arm debug interface v5. It is a standard interface for ARM processor-based devices. It is useful where limiting pin count is crucial. SWD uses a bus called Debug Access Port (DAP). DAP has one master (Debug Port - DP) and one or multiple slaves (Access Ports - APs). The image below shows the SWD architecture (Source - Reference).


DAP
internal bus
JTAG-AP
JTAG
Existing scan chains in core
AHB
System Peripherals
SWJ-DP
AHB-AP
AHB
System AHB matrix
Memory block
APB
RealView
APB-AP
APB
APB-mux
Debug APB
Debug Peripherals

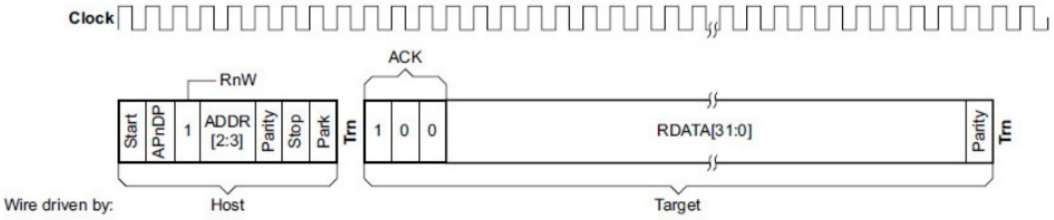
SWD architecture

The external debugger connects to DAP using the debug port (DP). There are three debug port interfaces to access the DAP.

- JTAG Debug Port (JTAG-DP) - It uses standard JTAG interface and protocol
- Serial Wire Debug Port (SW-DP) - It uses SWD protocol
- Serial Wire / JTAG Debug Port (SWJ-DP) - It can use either JTAG or SWD to access the DAP. In this, TCK and TMS JTAG signals are reused as SWCLK and SWDIO signals, respectively. For switching from one interface to the other, a specific sequence has to be sent.

SWD transaction has three phases.

- 8-bit Request phase sent from the host
- 3-bit ACK (Acknowledgement) phase sent from the target.
- Data phase where up to 32 bits are transferred from/to the host with a parity bit. When the data direction has to be changed, TRN (Turnaround) cycle is sent. The image below shows the SWD read transfer (Source - Reference).


Clock
RnW
ACK
Start
APnDP
1
ADDR [2:3]
Parity
Stop
Park
Trn
1
0
0
RDATA[31:0]
Parity
Trn
Wire driven by:
Host
Target

In the upcoming sections, we will discuss the attack scenarios and methods.

2. Possible Attack Scenarios

Getting access to the JTAG/SWD interface on the hardware opens many possibilities for an attacker to break into the device. The following are the possible attack scenarios:

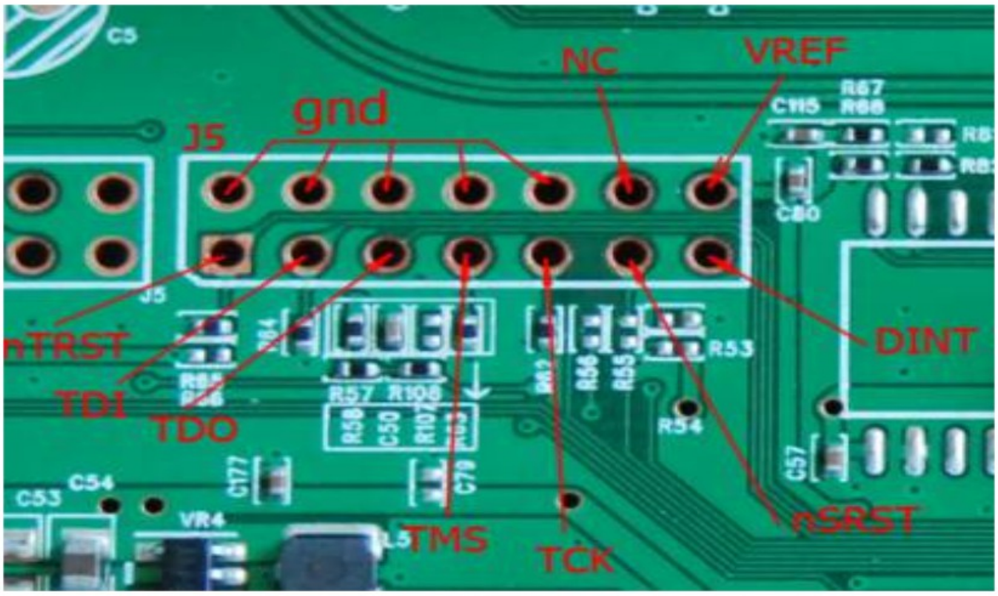
- Attackers get access to the controller's internal memory leading to the manipulation of the register values. Manipulating the internal registers can have varying effects. For example, in some cases, even if a controller read-protection is implemented, the adversary can manipulate the register value, and there is a possibility of bypassing the protection implementation. Similarly, the attacker could also change settings and bypass protection related to the bootloader and other critical registers that the developer might have locked for security aspects.
- Having access to the hardware with the JTAG/SWD interface gives the attacker the possibility of debugging the system. It may help him/her to unearth more vulnerabilities and perform attacks on the device.
- Another most significant advantage from the attackers' perspective is that the access to JTAG/SWD debug interface opens up the possibilities to extract the firmware from the device, patch the firmware, and re-flash the modified vulnerable/malicious firmware back into the device. Getting access to the firmware opens up vast possibilities for an attacker to catch and exploit other vulnerabilities.

3. Performing the Attack

We first need to identify the respective port pins on the hardware to interact with the device via JTAG/SWD port. Below are few images showing how most JTAG port test points/pinouts on the hardware may look like.

[redacted]

(Source - here)



(Source - here)

3. Performing the Attack

We first need to identify the respective port pins on the hardware to interact with the device via JTAG/SWD port. Below are few images showing how most JTAG port test points/pinouts on the hardware may look like.

3.1 Recon

You can refer to the 13th Chapter for a better understanding of the hardware recon.

Case 1: You do not have the actual hardware, but you know the FCCID number of the device.

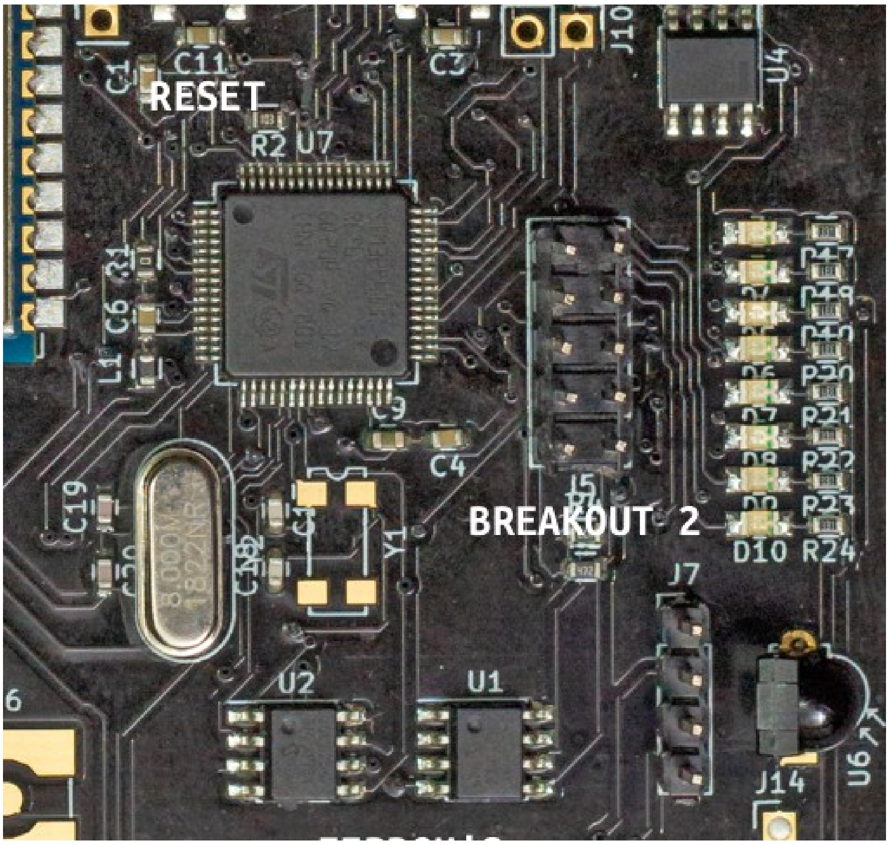
Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. You may get a hint for devices with JTAG/SWD interfaces on the hardware seeing the internal image. Yayy!! Once you know that you have the attack surface, you can get/purchase that device and perform further required steps to attack the device.

Case 2: You got access to the hardware.

Once you have the physical hardware in your possession, the first step should be to perform reconnaissance. Inspect each test point and chips present on the printed circuit board (PCB) to see if you can get a JTAG/SWD interface.

As discussed above, for JTAG, we need to identify four signal pins (TCK, TDI, TDO, TMS) and Vcc, GND pins. For SWD, we need two signal pins (SWDIO, SWCLK) and Vcc, GND pins. Few ways in which you can identify these interface pins on the device are mentioned below :

1. Manual Identification - Identify the microcontroller used in the device, take out its datasheet, and identify the microcontroller's JTAG/SWD pins. For example, we will take the example of the EXPLIoT DIVA board.



The microcontroller used in the diva board marked as U7 on the PCB is **STM32F411x**, we can search its datasheet for the pins having JTAG/SWD interface connection. The image below shows that section of the datasheet.

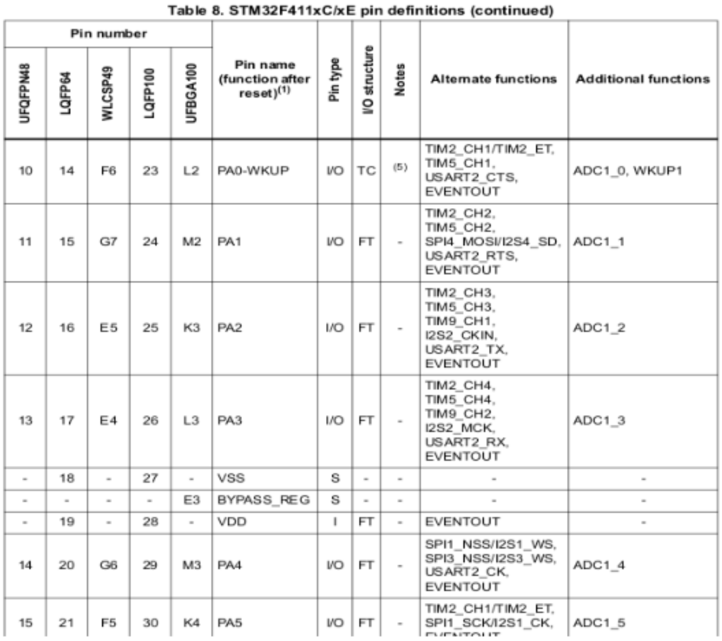


Table 8. STM32F411xC/xE pin definitions (continued)

Pin number					Pin name (function after reset)[1]	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	LQFP64	WLCSP49	LQFP100	UFBGA100						
10	14	F6	23	L2	PA0-WKUP	I/O	TC	(5)	TIM2_CH1/TIM2_ET, TIM5_CH1, USART2_CTS, EVENTOUT	ADC1_0, WKUP1
11	15	G7	24	M2	PA1	I/O	FT	-	TIM2_CH2, TIM5_CH2, SPI4_MOSI/I2S4_SD, USART2_RTS, EVENTOUT	ADC1_1
12	16	E5	25	K3	PA2	I/O	FT	-	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, USART2_TX, EVENTOUT	ADC1_2
13	17	E4	26	L3	PA3	I/O	FT	-	TIM2_CH4, TIM5_CH4, TIM9_CH2, I2S2_MCK, USART2_RX, EVENTOUT	ADC1_3
-	18	-	27	-	VSS	S	-	-	-	-
-	-	-	-	E3	BYPASS_REG	S	-	-	-	-
-	19	-	28	-	VDD	I	FT	-	EVENTOUT	-
14	20	G6	29	M3	PA4	I/O	FT	-	SPI1_NSS/I2S1_WS, SPI3_NSS/I2S3_WS, USART2_CK, EVENTOUT	ADC1_4
15	21	F5	30	K4	PA5	I/O	FT	-	TIM2_CH1/TIM2_ET, SPI1_SCK/I2S1_CK,	ADC1_5

Depending on the IC package that is used in the device (here it's LQFP64), we identify the JTAG port pins on the controller. Then, we set the multimeter in continuity mode as in the image below:

Then, we put one probe on the JTAG port pins on the controller and the other probe on the test points/ headers pins on the PCB suspected to be JTAG port pins. This test is repeated until pins are identified. The procedure for identifying SWD pins is the same. Manual identification is undoubtedly going to be hectic, but no worries; we have automation tools with us to solve this problem:

2. Automated Identification- Many available tools can help us scan and identify the JTAG/SWD pins on the hardware.
   - EXPLIoT Bus Auditor - It supports JTAG, SWD, I2C, UART. It has an adjustable target voltage. Its demo example in the case of JTAG and SWD interface can be understood from the 14th chapter of this e-book.
   - JTAGEnum - It's an open-source project based on Arduino. It does not support UART, and also it does not have the adjustable voltage. It is cheaper to use. Before connecting pins to Arduino, make sure to adjust the voltage levels. You can refer to the blog here to understand how to identify JTAG pins using JTAGEnum.
   - JTAGulator - It supports JTAG, UART, SWD. It has an adjustable target voltage. You can refer to the blog here to understand how to identify JTAG pins using JTAGulator.

NB : Before connecting any external device to the target board, make sure to adjust the voltage levels and make the Ground common for both the attacking/scanning and the target devices.

Yayy!! Once the JTAG/SWD interface pins are identified on the hardware, it is time to break into the device.

3.2 Sniff the communication

Tools like Logic Analyzer can be used to sniff the communication between the device having the JTAG/SWD interface and the controller. The logic analyzer would show the TCK, TDI, TDO, TMS signal lines of the JTAG port or SWCLK , SWDIO lines of SWD port. Softwares like Saleae Logic Analyzer, PulseView have the feature to detect these signals that directly shows you the decoded data w.r.t the respective JTAG/SWD interface. Sniffing communication can help to decode the IDCODE and other sensitive information.

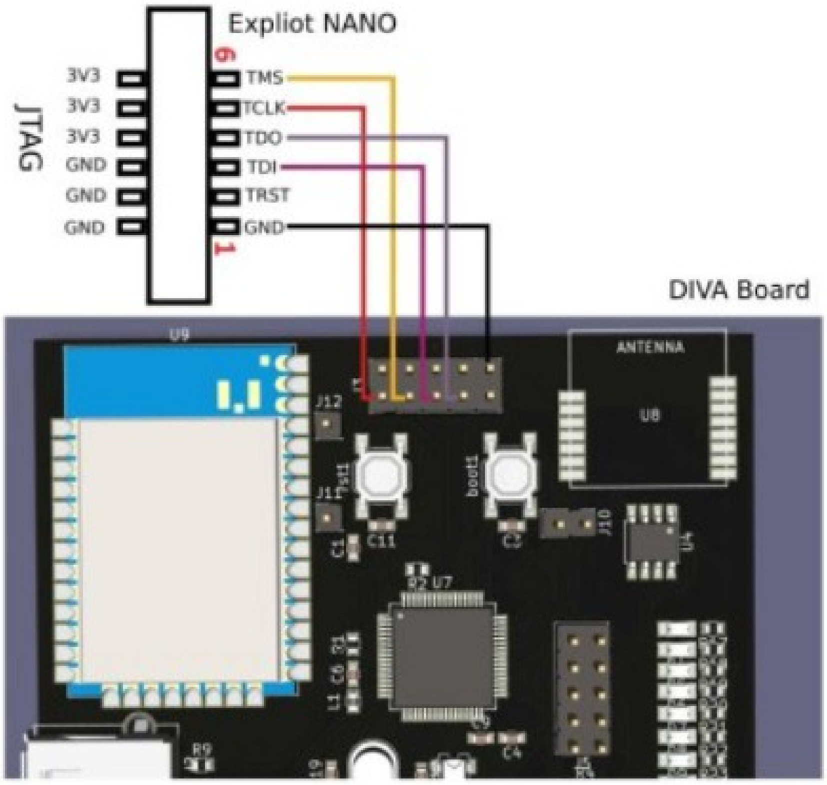
3.3 Interfacing

To communicate with the JTAG/SWD interface on the device, we need a protocol adapter and a software that can communicate over JTAG/SWD via the adapter with the device. Keep note of an important point again, " Before connecting any external device/ adapter to the target board, make sure to adjust the voltage levels and make the Ground common for both the attacking/scanning and the target devices." Various adapters are available for connecting with JTAG/SWD interface. A few of them are mentioned below.

- EXPLIoT Nano
- Bus Pirate
- Shikra

For communicating with JTAG/SWD interface with the host machine via these protocol adapters, we need something that can take instructions from the user via the host machine and send the corresponding low-level instructions to the respective JTAG/SWD interface via the protocol adapter and vice-versa. **Openocd: Open on-Chip Debugger** is a debugging software that provides on-chip programming and debugging support with a layered architecture of JTAG/SWD interface and TAP support. It supports Debug target (e.g., ARM, MIPS): single-stepping, breakpoints/watchpoints, etc. It has support for a variety of chips, interfaces, and targets. It also opens GDB and telnet server. You can perform GDB debugging on the hardware or communicate over telnet using openocd commands. To communicate with the respective chip and the interface, we need to provide the correct configuration files w.r.t the chip and the interface we are using. Below, we will show JTAG interfacing using EXPLIoT Nano on the target EXPLIoT DIVA board.

After identifying the JTAG pins by any of the scanning tools/ methods discussed above, connect the JTAG pins TCK, TMS, TDI, TDO with the corresponding JTAG pins on the adapter (here, EXPLIoT Nano as per its datasheet/user manual). The connection image is shown below.

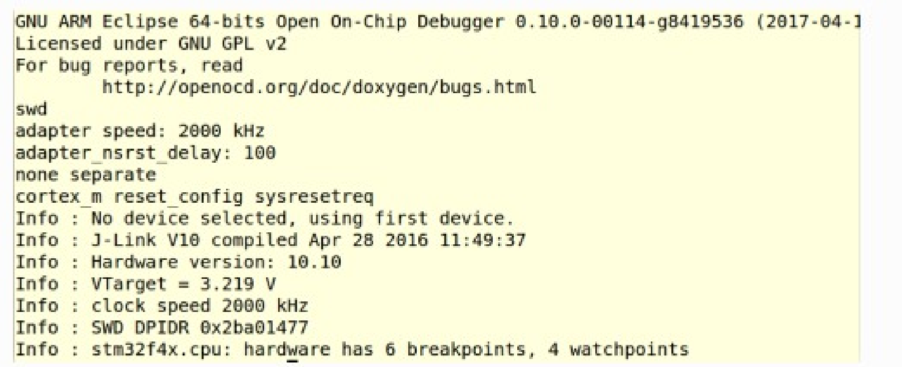

Expliot NANO
9
3V3
3V3
3V3
GND
GND
GND
TMS
TCLK
TDO
TDI
TRST
GND
1
JTAG
DIVA Board
U9
J12
J11
rst1
boot1
ANTENNA
U8
J10
U4
C1
C11
C2
R2 U7
R9
C4

Now, to communicate with the JTAG interface, we need to launch the

Openocd, `$ openocd -f expliot_nano_jtag.cfg` .

> Note: We pass configuration files to openocd to make it aware of the target microcontroller and the debug adapter used. For a different microcontroller, adapter configuration, you might need to choose the corresponding configuration files. More on openocd configuration files can be read [here](http://openocd.org/doc/html/Config-File-Guidelines.html)*.

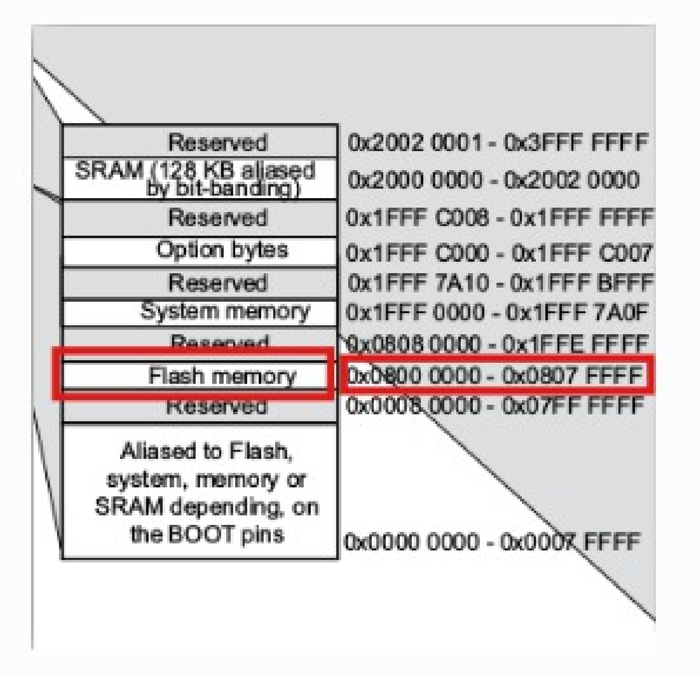
The communication with the interface starts as in the image below.



```
GNU ARM Eclipse 64-bits Open On-Chip Debugger 0.10.0-00114-g8419536 (2017-04-1
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
swd
adapter speed: 2000 kHz
adapter_nsrst_delay: 100
none separate
cortex_m reset_config sysresetreq
Info : No device selected, using first device.
Info : J-Link V10 compiled Apr 28 2016 11:49:37
Info : Hardware version: 10.10
Info : VTarget = 3.219 V
Info : clock speed 2000 kHz
Info : SWD DPIDR 0x2ba01477
Info : stm32f4x.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Now, in another terminal, we will open telnet and perform different operations over the JTAG interface. The link here describes in detail the commands for performing different operations including, memory read, write, dump, etc.
So, for example to extract the firmware the command is , `dump_image flash_dump.bin <address> <flash memory length>`

We first need to identify the address and length of the flash memory of the chip that we are using. As discussed above, the EXPLIoT DIVA board has an **STM32F411x** microcontroller. We check its datasheet , we identified the flash memory address at **0x8000000** and length, **0x807FFFF – 0X8000000 = 7FFFF** as shown in image below.

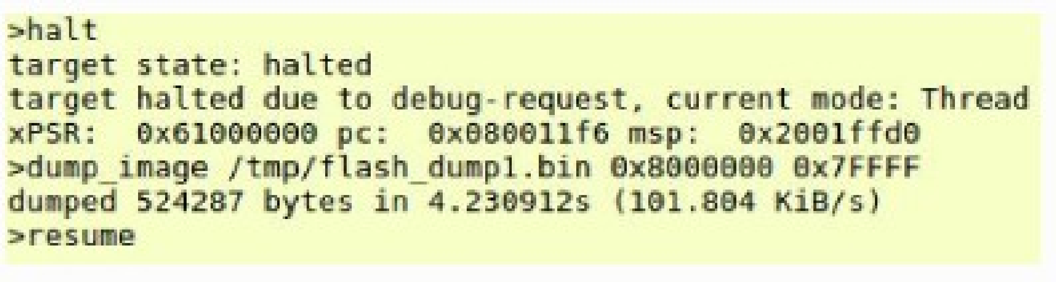


Reserved	0x2002 0001 - 0x3FFF FFFF
SRAM (128 KB aliased by bit-banding)	0x2000 0000 - 0x2002 0000
Reserved	0x1FFF C008 - 0x1FFF FFFF
Option bytes	0x1FFF C000 - 0x1FFF C007
Reserved	0x1FFF 7A10 - 0x1FFF BFFF
System memory	0x1FFF 0000 - 0x1FFF 7A0F
Reserved	0x0808 0000 - 0x1FFE FFFF
Flash memory	0x0800 0000 - 0x0807 FFFF
Reserved	0x0008 0000 - 0x07FF FFFF
Aliased to Flash, system, memory or SRAM depending, on the BOOT pins	0x0000 0000 - 0x0007 FFFF

And via the telnet, we send the following commands,

- halt
- dump_image flash_dump.bin 0x8000000 0x7FFFF
- resume

It successfully extracts the firmware from the device, as shown below.



```
>halt
target state: halted
target halted due to debug-request, current mode: Thread
xPSR:  0x61000000 pc:  0x080011f6 msp:  0x2001ffd0
>dump_image /tmp/flash_dump1.bin 0x8000000 0x7FFFF
dumped 524287 bytes in 4.230912s (101.804 KiB/s)
>resume
```

Thus, various openocd commands can be used to perform various operations, including manipulating the register value, patching the firmware, etc. A similar approach would be for the SWD interface, we just need to connect with the respective SWD pin, and corresponding SWD transport should be mentioned in the configuration file. Thus, once we are successful in interfacing and communicating with the JTAG/SWD ports, it opens the possibilities for various attack scenarios, as discussed above.

4. Conclusion

In this chapter, we learned about the JTAG, SWD interface, the possibilities of attack scenarios, and the methods to attack. We hope you enjoyed and got some valuable information out of it. These tools and attack methods would give you a basic understanding of what to do when you find a JTAG/SWD interface on the hardware and play around with it to attack some devices.

Side-Channel Analysis
Chapter 19: Introduction To Side-Channel Attacks (SCA)

In the last few chapters of this e-book, we discussed hardware attack surfaces like SPI, I2C, UART, and JTAG. With an increase in the security concerns and awareness, proper countermeasures are implemented in some devices to avoid easy access to the attacker via these attack surfaces. Unfortunately, even these countermeasures against hardware attacks cannot assure a secure system. This chapter will give a basic overview of one of the most famous hardware attacks called the Side-Channel Attacks (SCA). This chapter is an introductory, conceptual overview of SCA.

1. Introduction

Side-Channel Attacks (SCA) exploit the information leakages in the system. The leakages can be related to timing, power, electromagnetic signals, sound, light, etc. SCA is a non-invasive and passive attack, i.e., to perform this attack, we don't need to remove the chips to get direct access to the device's internal components or actively tamper any of its operations. This attack can be performed by observing, collecting, and then analyzing the information leakages in the device during processing. These attacks can be used to retrieve any sensitive information from the device. They are most commonly used to target cryptographic devices. Though attacking such devices with a very large keyspace might not be feasible via brute-force, attackers can break these so-called secured systems via SCA. Instead of targeting the standard cryptographic algorithms, SCA targets their implementation on the physical devices to recover the secret parameters by measuring and analyzing the leaked information like power analysis, timing analysis, electromagnetic analysis, etc.

2. Types of side channel attacks

2.1 Power Attacks

Depending on the data or the code instruction being processed in the device, its power consumption varies. In this attack, the varying power consumption is measured and analyzed to extract the device's sensitive information or keys. It can be used to break the implementation of cryptographic algorithms like AES-256 in comparatively less time and at less cost.

Internally, the devices have integrated circuits that consist of logic gates that are internally made up of transistors. Depending on the charge applied and removed on the transistors' gate, the current flows to or from the gate, which consumes the observable power and leaks the internal characteristics of the operating algorithm. To measure the power in the device's circuit, a small resistor of approximately 50 ohms, is inserted in series with the power or the ground input. The voltage difference (V) across the resistor is calculated, that when divided by the resistance value®, gives the current(I) as per Ohm's law, V = IR. Depending on the clock frequency, equipment like digital oscilloscope, hantek, chipwhisperer, etc. can be used to sample the voltage differences and collect the trace used for further analysis and attacks. There are a few open-source frameworks like chipwhisperer, SCARED by eShard that can be used for the analysis.

2.1.1 Simple power analysis (SPA)

SPA is the visual analysis of the collected power trace while an operation is being performed to identify it. It is based on the fact that the power consumption at a particular instant of time is the function of operation being carried out by the device. It is targeted to work with fewer traces. This can be used to extract the key or some sensitive information. The power analysis using SPA can give the characteristic of the kind of algorithms being operated, e.g., the square and multiply operations, the number of rounds of block ciphers, etc. This attack involves comparatively more manual analysis. If the device's power consumption can be randomized internally by adding noise or including dummy clock cycles, then extracting the secret information via SCA gets difficult.

2.1.2 Differential power analysis (DPA)

DPA is more powerful than SPA as it also includes statistical analysis. It is based on the fact that the power consumption at different instances of time for the same operation depends on the data being processed. Though there are different types of DPA, the Difference of Means (DoM) is one of the simple and effective methods to perform it. In this case, it is assumed that the power consumption correlates to the specific bit of an internal register of the hardware that the attacker is not aware of. The portion of the key is guessed. Based on the guessed portion, the target bit is calculated. The trace is then divided into zero-bin and one-bin depending on whether the target bit is 0 or 1. The mean of all the traces in the zero-bin and the one-bin is calculated, the difference of which gives the DoM. Finally, the correct key guess and the wrong key guess are identified by correlating the power consumption between the guessed and the real power trace. As this attack does not require the attacker to have the system's internal knowledge, it is a black-box attack.

However, it is expensive in terms of the number of traces to be collected. It also has the problem of "ghost peaks," i.e., sometimes one can get a strong peak in the statistical graph even for the wrong guesses. More detailed info about DPA can be read from the research paper titled, "Differential Power Analysis," by Paul Kocher, Joshua Jaffe, and Benjamin Jun.

2.1.3 Correlation Power Analysis (CPA)

Research works in SCA areas later came out with the Correlation Power Analysis (CPA) method, which was found to be more effective than the DPA. The research paper, “Power Analysis Attacks to Cryptographic Cir-cuits: a Comparative Analysis of DPA and CPA” shows the comparative analysis between CPA and DPA. It requires a lower number of power traces w.r.t DPA and also avoids the problem of ghost peaks. CPA is implemented by using the power models : Hamming Weight or Hamming Distance. Hamming weight is linearly related to power consumption. So, instead of computing average power consumption for many traces, the device’s power model is produced that is correlated with the actual power consumption of the device using Pearson correlation. More detailed info about CPA can be read from the research paper titled, “Correlation Power Analysis with a Leakage Model.

2.2 Electromagnetic (EM) Attacks

These attacks exploit the electromagnetic emissions from the device during operation. Performing this attack needs an electromagnetic measurement probe. It’s suitable even for higher frequency signals.

As discussed before, under the power analysis attack, the devices have integrated circuits; within the integrated circuits, there are transistors; the flow of current within them creates the electromagnetic field. Thus, as the power is leaked during operation, similarly, there are electromagnetic leakages. The measurement and analysis of these signals give the attacker the possibility to extract the secret information/keys. Similar to power analysis, SPA and DPA can be performed on electromagnetic traces. EM-based SCA is preferred in most cases because it does not require any manipulation in the circuit by inserting a shunt resistor as in the power analysis. Though in terms of tools, it gets expensive. However, more research is being done to develop efficient and low-cost solutions to perform EM-based SCA. The research paper titled, “SCNIFFER: Low-Cost, Automated, Efficient Electromagnetic Side-Channel Sniffing” proposes a low-cost platform for this purpose.

2.3 Timing Attacks

In certain implementations of the device operation, the execution timing is dependent on the data inputs based on the different execution paths. The attacker exploits this execution timing difference to either guess or extracts the sensitive data. The research paper on “Timing Attacks on Implementations of Diffie-Hellman,RSA,DSS,and OtherSystems” by Paul C.Kocher shows the timing attacks that have been done on the cryptographic implementations to extract the secret key.

2.4 Cache Attacks

In this attack, the attacker exploits the information leaked during the cache access because the cache hit takes comparatively less time w.r.t cache miss. Sometimes, it is used with the timing attacks to determine the cache memory access timing. Though the accessed data cannot be seen, the attackers can extract the information by knowing the memory area being accessed. Meltdown and Spectre is one of the most severe types of cache and timing-based side-channel attacks on the Intel hardware that leaked the memory contents of the process and the operating system as well.

2.5 Differential Fault Attacks

These attacks involve extracting the keys/sensitive information by generating faults in the system. They can also exploit the fact that the device could be prone to faults due to higher design complexity and incomplete verification. Faults could be created by varying the voltage, clock glitching, etc.

More on fault attacks would be discussed in our next chapter. In this attack, the operation is performed, one with fault and the other without fault. Both are then analyzed and compared to identify the operation corresponding to the fault.

2.6 Acoustic Attacks

These attacks exploit the sound produced during the computation to extract the secret information. More on it can be read in the research paper titled, “RSA Key Extraction via Low - Bandwidth Acoustic Cryptanalysis”

3. Why side channel attacks are so alarming

DThere is a lot of effort and investment being made to make the critical system secure, especially hardware security. It is now more focused on implementing hardware-based encryption. Ranging from the implementation of secure hardware-based cryptographic devices using Trusted Platform Modules (TPM), implementing secure boot, etc., a lot of work is being done to ensure system security by securing the hardware along with the software. Attacks like SCA make it challenging towards the secure development of these devices. Varying products like our personal computers, smart cards, etc use these cryptographic devices, but the flaw in its implementation can open opportunities for SCA attacks.

SCA is also used to target deep neural network models. More about it can be read in the paper titled, “Open DNN Box by Power Side-Channel Attack”. Though on critical devices, like smart cards, proper countermeasures might be implemented, but SCA has become more dangerous for IoT devices, as these devices have resource constraints. Protection against these SCA attacks is neglected mostly on the IoT devices. In these devices, low power processors are used for running crypto operations in order to save energy.

Running unprotected crypto operations on generic processors makes it easier for the attacker to perform these attacks. As these attacks are more at the CPU level, they can put critical infrastructure and devices at significant risk that software patching cannot solve. With the increase in sophisticated tools and methods, these attacks have become more reachable. Read more about its impact on cryptographic modules by YongBin Zhou, DengGuo Feng.

4. Possible countermeasures for side channel attacks

Though it is practically difficult to eliminate these SCA attacks, they can be made harder to achieve. A few countermeasures could be taken to make it harder for the attacker to perform these attacks.

- On the hardware side, the use of physically unclonable functions (PUFs) and proper shielding from leaking electromagnetic radiation should be preferred.
- On the implementation side, the device's operation should be independent of data, i.e., the number of clock cycles should be either uniform or randomized to avoid operation timing leakages.
- Error detection should be implemented to identify if any faults occur in the circuit.
- The keys should not be reused on different products.
- The power consumption in the device should be independent of operating instructions. One of the techniques used is to use the differential signals, where power spikes are not seen, or by masking or adding noise such that transition from 1 to 0 or 0 to 1 cannot be leaked.
- Secure programming pattern should be adopted as mentioned in the white paper by Riscure
- After the development, the device must go through tests against SCA attacks. Some resources for performing these tests include, "A testing methodology for SCA, written by Gilbert Goodwill, et al.", "Welch's T-testin Side-Channel Security Evaluations"

Various other research works are done and ongoing to implement more effective countermeasures against SCA attacks for their mitigation.

5. Conclusion

In this chapter, we went through a basic overview of Side-Channel Attacks and its types. We hope this gives you an overall understanding of it and the related severity of such attacks.

Fault Injection
Chapter 20: Introduction To Fault Injection Attack (FI)

In this chapter, we will give a basic overview of Fault Injection (FI) attacks. This chapter is an introductory, conceptual overview of FI.

A Fault injection (FI) attack is a physical attack on the device to inject the fault in the system deliberately to change its intended behaviour. It can bypass system security features, change system behavior to accomplish malicious intents, or extract the secret information, key, or even firmware by analyzing the erroneous outputs. There are various ways to inject fault including, voltage glitching, clock glitching, laser injection, electromagnetic (EM) injection, etc. When we say a glitch, it means injecting a short-lived fault in the system, i.e., creating a sudden change in the system that's not part of its normal operation. This may lead to brick the device, but an analyzed and target glitch with precision can create a potential security threat. It is an active attack. It is performed during the system's ongoing operations and sometimes invasive when the chip needs to be decapsulated, and electrical contact with the chip is required. Many research has been done using this technique to bypass secure boot, bypass the security of crypto wallet, hack casino slot machine, find various vulnerabilities in ESP32, cryptanalysis to extract the secret key of the implemented cryptographic algorithm. These attacks can have severe impacts as they can altogether bypass the device's security mechanism if successful.

1. Fault injection techniques

There are various ways in which faults could be injected into the system. It could be either via the hardware or the software targeting the logic faults. As embedded devices have a relatively small codebase, software fault injection might not provide a broader attack surface. Hardware fault injections have more severe impacts as they can completely break the security features and most devices are still not resistant to such attacks. These techniques are used to perform attacks like differential fault attack (DFA) that we had discussed in the previous SCA Chapter. Here, we will discuss a few techniques used for fault injection via the hardware.

1.1 Clock glitch

This technique involves the tampering of the system clock to change its intended behavior. The integrated circuits (ICs) internally have combinatorial logic blocks and flip-flops registers that share the same clock. The data is latched at either the rising or at the falling edge of the clock. It gets transferred between the two edges, and there is a propagation delay involved to transfer the data through logic blocks. For the circuit to operate correctly, the clock's time period should be greater than the max propagation delay plus some offset timings. Tampering with the clock signal affects the clock period that can potentially affect the device's ongoing operation. The attacker should have direct control over the clock line to change the clock length, i.e., the clock glitching attack cannot work for the devices with internal oscillators.

By introducing short pulses in the circuit, the attacker can inject the fault when any specific instruction is being executed to bypass it. If this fault is injected precisely at the right moment with accuracy, it can even attack the cryptographic implementations, skip the instructions to bypass the security checks, etc.

1.2 Voltage glitch

This technique changes the system's intended behavior by manipulating the voltage at a specific instance to inject the fault. Though the voltage glitching technique is low-cost, the arbitrary change in the voltage may result in unpredictable fault states but might not lead to an advanced exploitable attack. With proper analysis and a well-timed glitch or device, underpowering can even attack the cryptographic implementation. The properly-timed glitch on the power line can make the device skip specific instructions that can be exploited to bypass security features. Since the Vcc pins of the chip are mostly accessible, it becomes relatively easier to identify the attack point.

However, to perform the exploitable attack, the correct settings and the proper external glitch circuit gets tricky to achieve sometimes. It is because when we try to change the voltage level at the target power line, there are other components like decoupling capacitors and voltage regulators that do not allow us to keep the setting accurate and constant.

A paper titled, "Fault Injection using Crowbars on Embedded Systems" describes a novel way to use this technique. Another, a very high precision attack using voltage and temperature manipulation has been proposed in one of the research papers titled, "Precise Fault-Injections using Voltage and TemperatureManipulation for Differential Cryptanalysis"

1.3 EM glitch

This technique is used to inject faults in the device via an electromagnetic field. This is one of the most used techniques for fault injection as it is mostly non-invasive. For more efficiency, it is preferred to depackage the chip by removing its plastic package. In this case, a high or medium voltage pulse generator is used to produce an EM pulse to inject the transient fault in the system. The proper positioning of the EM probe is required for this technique. For this purpose, the X-Y table is used to precisely position the EM probe above the target chip's surface. Researchers at Raelize discovered various flaws in ESP32 using EM fault injection. More detailed information can be found on Raelize's website. The attacks may result in secure boot bypass, bypass flash encryption, execution on arbitrary code, access to flash memory contents, etc.

1.4 Optical injection

This technique involves the illumination of transistors in the target device that makes it conduct and inject faults in the system.

A research paper titled, "Optical Fault Induction Attacks" by Sergei P. Skorobogatov and Ross J. Anderson discusses performing optical attacks on secure microcontrollers and smart cards without using expensive laser equipment via a semi-invasive approach. Compared to the noninvasive attacks, invasive attacks, though expensive but very powerful to resist. The optical injection using a laser or high energy light source like a UV lamp is done by decapsulating the chip using acid for accuracy and precision. These attacks can even reset the microcontroller's internal protection fuses, wipe the particular part of the data in the memory, attack EEPROM and Flash memory, break cryptographic implementation, etc.

2. Examples of fault injection attacks

Differential fault attacks use any of the fault injection techniques, as mentioned above, to break the device security feature. When combined with power analysis, machine learning, etc., these techniques result in more powerful attacks. Many fault injection attacks have been done by researchers on many cryptographic devices like smart cards. Glitching attacks can break noncrypto devices' security, including the execution of arbitrary code on the device, attack on xbox 360, etc. Apart from general tools like probes, oscilloscope, there are specific fault injection tools and techniques available to perform these attacks, including chipwhisperer, chipshouter, Riscure FI tools, Raiden, glitchsink, etc. A few examples of fault injection attacks are discussed below.

CVE-2019-15894

- Espressif ESP32: Bypassing Secure Boot
- An attacker can bypass the Secure Boot verification at the startup of the ESP32 CPU via fault injection.
- Leads to the execution of unverified code from flash.
- It was identified that if the flash encryption is enabled, this attack could be mitigated. Hence the flash encryption was then later implemented.
- Reference: Espressif Advisory, CVE Mitre

CVE-2020-15048

- Espressif ESP32: Bypassing Flash Encryption
- An attacker can bypass the Secure Boot and Flash Encryption physical security features of the ESP32 CPU via fault injection.
- Leads to the execution of unverified code despite both the secure boot and flash encryption implementation.
- Reference: Espressif Advisory, Raelize

CVE-2020-13468

- Gigadevice GD32F130 devices: Debug interface permissions escalation
- An attacker can exploit the insufficiently physically protected inter-IC bonding wires in Gigadevice GD32F130 devices via fault injection to escalate the debug interface permissions.
- Leads to the extraction of firmware from the devices despite debugging protection implementation, reads the protected flash memory, performs arbitrary modifications on the system, etc.
- Reference: Research paper, CVE Mitre

CVE-2020-15808

- STM32 USB Device Library: Buffer overflow vulnerability exploit
- The buffer overflow vulnerability in the CDC communication interface code exploited using fault injection.
- Leading to the access of sensitive information, keys, obtaining firmware, etc., with the properly crafted USB packets.
- References: Black Hat Asia 2020, Grzegorz Wypych, Raiden

3. Countermeasures

There is much ongoing research work to find the countermeasures to mitigate fault injection attacks. Resistance to invasive fault attacks, where the chip is decapped, and the passivation layer is removed to bypass the shielding, gets comparatively more difficult. A few of them include:

- Proper shielding to mitigate EM-based attacks, making chip die resistant to UV light-based attacks, and proper physical hardening.
- Implementing proper fault detection mechanisms like error detection codes, external error checking circuits, etc., to detect the fault immediately and abort the further device operation.
- Implementing the resistance against the side-channel attacks as discussed in the SCA Chapter.
- Implementing the redundancy for the critical control signals, i.e., implementing the control signals more than once that behave differently to mitigate glitching effects.

Since these are physical attacks at the CPU level, they have high potential risks on the devices. However, these countermeasures cannot completely stop the attacker from performing these attacks but can make it more challenging for them to perform these attacks.

4. Conclusion

In this chapter, we discussed the basic conceptual overview of fault injection and its techniques. We looked at a few attack cases and countermeasures. Hope, you get some idea about the different impact these attacks can have and how the increasing possibilities of these attacks that are comparatively difficult to resist is a threat to the system.

Reference :

Chapter 16 : Hardware Attack Surface-I2C

- https://en.wikipedia.org/wiki/I%C2%B2C
- https://www.nxp.com/docs/en/user-guide/UM10204.pdf
- https://www.i2c-bus.org/
- https://learn.sparkfun.com/tutorials/i2c
- https://www.researchgate.net/publication/314629854_Hardware_Attacks_on_Mobile_Robots_I2C_Clock_Attacking

Chapter 17 : Hardware Attack Surface-UART

- https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
- https://cs140e.sergio.bz/notes/lec4/uart-basics.pdf
- https://media.blackhat.com/us-13/US-13-Kohlenberg-UART-Thou-Mad-WP.pdf
- https://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/

Chapter 18 : Hardware Attack Surface-JTAG, SWD

- https://blog.senr.io/blog/jtag-explained
- https://embeddedbits.org/2020-02-20-extracting-firmware-from-devices-using-jtag/
- https://en.wikipedia.org/wiki/JTAG
- https://www.corelis.com/educationdownload/JTAG-Tutorial.pdf
- https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture/serial-wire-debug
- https://www.cnblogs.com/shangdawei/p/4748751.html
- https://research.kudelskisecurity.com/2019/05/16/swd-arms-alternative-to-jtag/

Chapter 19 : Introduction To Side Channel Attacks (SCA)

- https://en.wikipedia.org/wiki/Side-channel_attack
- https://www.newae.com/embedded-security-101
- http://gauss.ececs.uc.edu/Courses/c653/lectures/SideC/intro.pdf
- https://www.rambus.com/blogs/side-channel-attack-targets-deep-neural-networks-dnns/
- https://whisperlab.org/introduction-to-hacking/talks/chipwhisperer
- https://troopers.de/downloads/troopers19/TROOPERS19_NGI_RT_Hardware_Side_Channels.pdf
- https://jochen-hoenicke.de/crypto/trezor-power-analysis/
- https://www.riscure.com/uploads/2018/04/Riscure_CheapSCAte_RSA.pdf
- https://paulkocher.com/doc/DifferentialPowerAnalysis.pdf
- https://www.iacr.org/archive/ches2004/31560016/31560016.pdf

- https://www.tandfonline.com/doi/pdf/10.1080/23742917.2016.1231523?needAccess=true
- https://eprint.iacr.org/2014/204.pdf
- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9201569
- https://scholarworks.umass.edu/cgi/viewcontent.cgi?article=1822&context=theses
- https://www.esat.kuleuven.be/cosic/publications/thesis-182.pdf
- https://csrc.nist.gov/CSRC/media/Events/Non-Invasive-Attack-Testing
- Workshop/documents/03_deBeer.pdf
- Hardware Security: Design, Threats, and Safeguards Book by Debdeep
- Mukhopadhyay and Rajat Subhra Chakraborty

Chapter 20 : Introduction To Fault Injection Attack (FI)

- https://www.riscure.com/fi/
- https://arxiv.org/pdf/1903.08102.pdf
- http://euler.ecs.umass.edu/research/bbkn-IEEEP-2012.pdf
- https://www.riscure.com/uploads/2017/10/Riscure_Whitepaper_Escalating_Privileges_in_Linux_using_Fault_Injection.pdf
- https://pdfs.semanticscholar.org/bd8b/724cf102a9e9f6347dfe67ff519220b2fbbd.pdf
- https://www.researchgate.net/publication/269270120_Precise_Fault-Injections_using_Voltage_and_Temperature_Manipulation_for_Differential_Cryptanalysis
- http://dl.ifip.org/db/conf/cardis/cardis2010/AgoyanDNRT10.pdf
- https://www.researchgate.net/publication/286813291_On_the_Effects_of_Clock_and_Power_Supply_Tampering_on_Two_Microcontroller_Platforms
- https://www.cl.cam.ac.uk/~sps32/ches02-optofault.pdf
- https://www.eeweb.com/fault-injection-attacks-a-growing-plague/
- https://pure.tugraz.at/ws/portalfiles/portal/23511745/Attacking_AUTOSAR_using_Software_and_Hardware_Attacks.pdf
- https://pure.tugraz.at/ws/portalfiles/portal/3103517/_CARDIS_Protecting_the_Control_Flow_of_Embedded_Processors_against_Fault_Attacks.pdf
- https://raelize.com/posts/raelize-fi-reference-model/

Part 6

MISCELLANEOUS

Chapters: 21

What to expect:

A showcase and breakdown of some of the most famous cyberattacks to help you understand various attack possibilities in an IoT ecosystem

Chapter 21: Famous IoT Attacks & Vulnerabilities

In the previous chapters of this e-book, we discussed the basic overview of IoT security, covering the introduction, attack surface, radio, firmware, IoT protocols, hardware attacks, etc. This is the last chapter of our journey through this IoT Security E-book.

We hope this e-book would be a good starting point for someone getting started in IoT security. In this chapter, we will discuss a few famous IoT attacks and vulnerabilities to conclude this e-book.

Though there are many IoT attacks with varying attack surfaces ranging from hardware, firmware, radio, etc. We have chosen a few of them for this chapter. Kindly note that the sequence in which these attacks have been mentioned here does not intend to convey anything.

1. Ripple20

JSOF Research labs discovered Ripple20. It's a series of 19 zero-day vulnerabilities in the low-level TCP/IP library developed by Treck, Inc. This affected library exists in various IoT and embedded devices, including medical, home, enterprise, networking, retail, aviation, power grids, etc., leading various devices at high risks. The risks include data theft, malfunctioning of industrial devices, patching with malicious code/ backdoor in the device, remote control, etc. Hence, the issue in one library created a ripple-effect for the millions of devices in the entire supply chain of various sectors. Out of the 19 vulnerabilities found, 4 are critical remote code execution with CVSS score greater than or equal to 9, 4 are having CVSS score greater than or equal to 7, 11 have various lower severity, DoS, info leaks, etc., and two were anonymously reported. The technical details on a few of its CVEs are mentioned below. Reference: JSOF website

CVE-2020-11896

- Remote code execution (RCE)
- For performing RCE, the prerequisites for the device is to support IP fragmentation and IP tunneling. If these prerequisites are not met, then the vulnerability remains DoS.
- An attacker can perform the RCE on the target device if he/she can send UDP packets to an open port of the device.
- Fixed on version 6.0.1.66

Reference: JSOF technical details

CVE-2020-11898

- Information leak
- The incoming IPv4 fragments over an IP-in-IP tunnel is not handled properly by the Treck TCP/IP stack.
- An attacker can leak memory from the heap leading to information leakage vulnerability.
- Fixed on version 6.0.1.66

Reference: JSOF technical details

CVE-2020-11901

- It consists of many critical client-side vulnerabilities in Treck's TCP/IP stack DNS resolver.
- The vulnerability is in the DNS resolver component of the network stack. It occurs from an incorrect DNS label length calculation.
- After the successful exploitation, an attacker can perform RCE to respond to a DNS query generated from an affected device.
- Fixed on version 6.0.1.66

Reference: JSOF technical details
More details about all the vulnerabilities can be read at JSOF Ripple20 Technical Section

2. Meltdown and Spectre

Meltdown and Spectre are the hardware vulnerabilities in processors to leak/steal information by abusing CPU data cache timing via side-channel analysis.

Meltdown attack exploits the side effects like leakages via timing differences and cache analysis of out-of-order execution features present in most of the modern processors. A more detailed working can be read from the Meltdown research paper mentioned under the reference. It bypasses the memory isolation security feature between the user applications and the operating system that ensures the kernel address range should be non-accessible and protected from user access. The adversary with non-privileged access can read the entire kernel memory through this attack, including any mapped physical memory without any permission. It was thus leaking critical information of the operating system and the other programs.

Spectre attack exploits the branch prediction and speculative execution feature of processors. These attacks trick the processor to speculatively execute instructions that should not have occurred during the correct program execution. The CPU states are reverted to maintain the functional correctness, which is called transient instructions.

The adversary can then leak the sensitive information via side-channel analysis from the processor's memory address space by influencing the transient instructions that are speculatively executed. There are three known variants of these vulnerabilities, variant 1: CVE-2017-5753 and variant 2: CVE-2017-5715 is grouped under Spectre attack, and variant 3: CVE-2017-5754is grouped under Meltdown. There are patches to prevent from the known exploits based on Meltdown and Spectre through software patches.

References: Meltdown: Reading Kernel Memory from User Space, Spectre Attacks: Exploiting Speculative Execution, Meltdown attack, IoTSF, Ars Technica, The Register, Google Project Zero

3. Mirai botnet attack on IoT devices

The IoT botnet attack exploits the insecure IoT devices with open Telnet ports and weak/default credentials to build a botnet. IoT devices located at inaccessible locations without the feature to be remotely patched become the victim of such attacks. Mirai is a malware that turns Linux based network devices into bots to form a large botnet army. It has been used to perform major Ddo attacks, including Brian Krebs' website, French web host OVH, Dyn cyberattack. It is used to convert the insecure IoT devices into bots. The infected devices then scan for other vulnerable IoT devices to infect them, thus becoming self-propagating. The creators made the source code for Mirai open source. Since then, it's used to perform attacks on many IoT devices.

References: csoonline, IoTSF, Wikipedia

4. Rolljam Attack

This attack, demonstrated by Samy Kamkar, targeted the key fobs of cars that are used to unlock the door remotely and for other purposes. He developed a gadget called "Rolljam," a radio device to perform this attack in the vehicle's radio range. When the door unlock button is pressed on the key fob, it transmits the code via the radio signal that is received by the car. The door gets opened if the code received matches the car's code. The key fobs use a rolling code security mechanism to send these codes where each time a new code is generated and old codes that are already used are discarded to avoid replay attacks by the attacker. But the rolljam attack brokes this security mechanism by recording and jamming the radio signal from the key fob. The car doesn't unlock because the signal is jammed. Hence the owner has to send the signal again. The second signal is also recorded and blocked, but this time the attacker unlocks the door by broadcasting the blocked first code. The adversary now also has the second sequence of recorded code that is not used by the car yet and hence not discarded.

The adversary can use it to unlock the door anytime later.

References: Wired article, Research paper titled, "Hold the Door! Fingerprinting Your Car Key to Prevent Keyless Entry Car Theft", Hackster.io article

5. Sweyntooth vulnerabilities

Sweyntooth is a collection of Bluetooth Low Energy (BLE) vulnerabilities discovered by a group of researchers at SUTD Asset Research group. As of now, it has a family of 18 vulnerabilities. The vulnerabilities are across the various BLE software development kits (SDKs) of major systems on chips (SoCs), exposing the BLE implementation flaws. Depending on the applications, it allows an adversary in the radio range to cause crashes, deadlock; buffer overflows, security bypass, etc. It affected various IoT devices having the vulnerable BLE stack from different vendors.

The CVEs related to these vulnerabilities include : CVE-2019-16336, CVE-2019-17519, CVE-2019-17061, CVE-2019-17060, CVE-2019-17517, CVE-2019-17518, CVE-2019-17520, CVE-2019-19193, CVE-2019-19195, CVE-2019-19192, CVE-2019-19196, CVE-2019-19194, CVE-2020-13593, CVE-2020-13595, CVE-2020-10061, CVE-2020-10069, CVE-2020-13594.

As mentioned in the Sweyntooth github repo, Zero LTK Installation CVE-2019-19194 is the most critical Sweyntooth vulnerabilities. It allows the adversary to bypass the latest Bluetooth pairing mechanism altogether. More detail about the other CVEs can be read from the links mentioned in the references.

Though many vendors have released the patch, as mentioned on the research page, there could still be various unverified devices with these vulnerabilities. To automate these testings, the researchers have released the open-source Sweyntooh Framework with proof of concepts.

References: SUTD Asset Research Group, SweynTooth: Unleashing Mayhem over Bluetooth Low Energy

6. Sweyntooth vulnerabilities

The vulnerability was found in the Philip Hue smart bulbs. It exploited the heap-based overflow in the implementation of the Zigbee wireless protocol while handling a long ZCL (Zigbee cluster library) string during the commissioning phase. It resulted in remote code execution in the bridge component that is responsible for sending the remote commands to the bulb over Zigbee protocol. To achieve this, the attacker has to first compromise the bulb with the malicious firmware (this was achieved a few years before by exploiting the vulnerability in the Zigbee Light Link (ZLL) protocol implementation. It can be referred to in the report titled,"IoT Goes Nuclear: Creating a ZigBee Chain Reaction"). The compromisation of the bulb with the malicious firmware makes it appear unreachable from the user control app. It tricks the user into thinking that there is just a fault. To reset the bulb, the user removes it from the app and instructs the bridge to discover the bulb. The bridge discovers the compromised bulb that then gets added to the network. Exploiting the Zigbee implementation vulnerability to trigger the heap-based overflow on the bridge enables the adversary to send malicious code (malware) to the bridge. The infected bridge connects back to the hacker, infiltrates the network using known exploits, and compromises other devices, including the target network's computer network. The vulnerability in a smart bulb leads to the compromisation of the complete target network. The corresponding CVE is: CVE-2020-6007. After the Checkpoint research team's responsible disclosure, the patch has been released. Still, the main issue is in the Zigbee protocol implementation that can attack the other IoT devices in similar ways.

Reference: Checkpoint research

7. Fault Injection Attack on ESP32

Researchers at Raelize have discovered various flaws in ESP32 using EM (Electromagnetic) fault injection attacks. The attack, as mentioned in CVE-2019-15894 bypasses the secure boot verification that was later patched by implementing flash encryption. Later, the researchers bypass even this flash encryption and thus the secure boot via fault injection. The corresponding CVE is CVE-2020-15048. These attacks lead to the execution of unverified code, escalating privileges on Linux, arbitrary control of the program counter (PC) register, etc. More detail about these attacks can be read in the previous chapters. ESP32 is low-cost and easy to use. It is used by many hobbyists and also in developing low-cost IoT products. Attacks like these can affect other chips used to develop IoT products that might not be resistant to such attacks.

Reference: Raelize

8. BLESA Attack

BLESA, i.e., Bluetooth Low Energy Spoofing Attack, exploits the BLE software stack implementation vulnerability. This vulnerability is related to BLE's reconnection procedure, during which the two previously connected devices reconnect. As per the BLE specification analysis, researchers found that the BLE specification does not mandate it to perform authentication during reconnection. The vendor implementation can take this reconnection authentication part optional despite following the specification hence prone to attack. These critical weaknesses identified in the BLE specification make the BLE devices vulnerable to spoofing attacks. This attack enables the adversary to impersonate the BLE device and provide the spoofed malicious data to the previously paired device. It was found that this attack applied to many BLE stack implementations, including Linux BlueZ, Android, and iOS. Though Apple assigned the CVE-2020-9770 to the vulnerability and fixed it, as per researchers, the Android on which this attack was made is still vulnerable. As many IoT devices have BLE implementation, this is a critical threat if the stack implemented inside them possess the same vulnerability.

References: Research Paper titled, "BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy", Zdnet Article

9. Bluetooth attack on Tesla Model X

The recent Bluetooth attack on Tesla Model X's keyless entry system showed that the security researcher named "Lennert Wouters" at Belgian university KU Leuven could break into the car in 90 seconds. The vulnerability in the key fob firmware update mechanism over Bluetooth was exploited, allowing the adversary to patch the key fob with the malicious firmware. It shows the lack of a code signing feature in the over-the-air firmware update mechanism of the key fob. The researcher revealed that a group of security vulnerabilities was found in the Tesla Model X car and its key fobs. These vulnerabilities can be combinedly exploited to unlock and steal the car. The malicious firmware in the keyfob was able to query the secure enclave chip responsible for generating the car's unlock code. The details about it can be read from the Wired article mentioned under the references.

Reference: Wired article

10. Amnesia:33

Amnesia:33 discovered by the Forescout research team is a collection of 33 vulnerabilities in four open-source TCP/IP stacks (uIP, FNET, picoTCP, and Nut/Net). The vulnerabilities include improper memory management, overflow flaws, lack of input validation, etc. The exploitation of these vulnerabilities could lead to memory corruption, remote code execution, perform DoS (Denial of service0) attacks, information leaks, etc. The TCP/IP stack is used in millions of IoT devices ranging from consumer IoT, industries, power grids, medical, access controls, servers, routers, the whole supply chain, etc. We also saw the Ripple20 vulnerabilities that were found in the TCP/IP stacks. As per researchers, more than 150 vendors and millions of devices are vulnerable to Amnesia:33. Thus, these vulnerabilities possess potential threats. There could be possibilities of the presence of vulnerable TCP/IP stacks in many code bases of various products leading to more such attacks.

References: Hackernews article, Forescout research lab, Blackhat

10. Amnesia:33

There are a lot more different attacks related to IoT. In this chapter, we picked up a few of them to show the various attack possibilities in the IoT ecosystem, ranging from hardware, firmware, radio, etc. We hope you got the idea of the threat on the IoT devices and why it is so essential to gear up IoT security.

References: Hackernews article, Forescout research lab, Blackhat

Reference :

Chapter 21 : Introduction To Fault Injection Attack (FI)

- Hackernews article
- Forescout research lab
- Blackhat
- JSOF technical details
- Meltdown: Reading Kernel Memory from User Space
- Spectre Attacks: Exploiting Speculative Execution
- Meltdown attack
- IoTSF
- Ars Technica,
- The Register,
- Google Project Zero

Summary

This concludes the end of the e-book, where we went over concepts like IoT Hardware, Protocols, Radio, Firmware, and other critical fundamentals to educate you with everything you'd need to get started with Practical IoT hacking.

If you're looking to delve into IoT security, check out our products at the EXPlIoT Website, which aim to equip you with the basic hardware and guides you'd need to get into IoT assessment.

We hope this e-book gave you valuable knowledge and insight into the world of IoT security.

Copyright © 2021 Payatu Consulting Pvt. Ltd. All Rights Reserved.